Universidade do Minho
Escola de Engenharia

Jorge Miguel da Silva Brandão Gonçalves

Development of tools for sentiment analysis in the Portuguese Language

November, 2022

Universidade do Minho
Escola de Engenharia

Jorge Miguel da Silva Brandão Gonçalves

Development of tools for sentiment analysis in the Portuguese Language

Master Thesis
Master in Informatics Engineering

Work developed under the supervision of:
Miguel Francisco Almeida Pereira Rocha
Vítor Manuel Sá Pereira

November, 2022

Acknowledgements

Firstly, I would like to sincerely thank my family for all their love, care and support, especially my parents Abel and Alice, my older brother José and my twin brother João. Without their love and encouragement, all these years would have not been possible.

Secondly, I would like to thank Dr. Miguel Rocha and Dr. Vitor Pereira for giving me the great opportunity to work with them and the Omnium AI team. I would like to praise the availability, support and expertise shown by both.

I would also like to thank them for integrating me with the Omnium AI Team giving me access to everything I needed.

I would like to thank Omnium AI members, especially Rubén Rodrigues, Nuno Alves and Fernando Cruz, for their unwavering help.

This thesis would not have been possible without the help of both Professors and the Omnium AI members patient guidance throughout the thesis.

I would like to praise *Universidade do Minho*, specifically *Departamento de Informática* and their teaching personnel for giving me the knowledge and the means throughout my degrees.

Finally I would like to thank all my friends for their help, care and support throughout these last 5 years.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

“Perfection is not attainable, but if we chase perfection we can catch excellence.” (Vince Lombardi)

Resumo

A Análise de Sentimentos é uma das áreas mais importantes na ciência da computação, nomeadamente no Processamento da Linguagem Natural. As suas aplicações vão desde a análise de produtos até à contenção do cyberbullying. A importância da análise dos sentimentos é inigualável, mas quando se trata de línguas menos faladas, o campo parece ficar para trás.

Neste contexto, Omnium AI propôs uma dissertação onde exploramos a Análise de Sentimentos para a Língua Portuguesa, com a intenção de criar uma nova ferramenta computacional. Esta dissertação vai examinar o campo da análise de sentimentos e o desenvolvimento do package Omnia. Este package é composto por ferramentas para a leitura de dados, o seu processamento e a criação de modelos Machine Learning (ML) e Deep Learning (DL) a partir dos dados lidos. Em específico, vamos concentrar-nos no desenvolvimento do package Omnia Text Mining, com o objectivo de criar ferramentas de pré-processamento e modelos de ML e DL para a análise de sentimentos para a língua portuguesa.

Esta dissertação vai criar uma abordagem para lidar com problemas de análise de sentimentos composta por um processo de recolha de dados, seguido de um passo de pré-processamento e acabando com o desenvolvimento de modelos de ML e DL. Esta abordagem será aplicada ao tópico do Covid-19. Após serem criados os modelos para os datasets relativos ao Covid, avaliamos os resultados para as diferentes combinações de métodos de pré-processamento e modelos onde apuramos que as Long Short Term Memory (LSTM)s e o HFAutoModel com o embedding Bert foram os melhores modelos. No geral, os modelos de DL e Autogluon obtiveram melhores resultados que os modelos de ML. Nos métodos de pré-processamento visualizamos que não existe uma *Pipeline* geral que possa ser utilizada para todos os casos.

No final, iremos discutir as conclusões que podemos retirar desta dissertação juntamente com uma secção de trabalho futuro, onde exploraremos os próximos passos possíveis para este projecto.

Palavras-chave: Deep Learning; Machine Learning; Text Mining; Sentiment Analysis

Abstract

Development of tools for sentiment analysis in the Portuguese Language

Sentiment Analysis is one of the most important areas in computer science, namely in Natural Language Processing. Its applications range from product reviews to cyberbullying containment. The importance of sentiment analysis is unprecedented, but when it comes to lesser-used languages, the field seems to be lagging behind.

In this context, Omnium AI proposed a dissertation where we explore Sentiment Analysis for the Portuguese Language with the aim of creating a new computational tool. This dissertation is going to delve into the sentiment analysis field and the development of the Omnia package. This package is composed of tools for reading datasets, processing them and creating ML and DL models from the data read. Specifically, we will focus on developing the Omnia Text Mining package, with aim of creating pre-processing tools and models for Sentiment Analysis (SA) in the Portuguese Language.

This dissertation creates an approach to tackle SA problems that involve a data gathering step followed by a pre-processing step and finishing with a model step where we develop different ML and DL models. This approach will be applied to a Covid-19 topic. From this approach, we obtained two datasets, from which we created ML, DL and Autogluon models. After creating the models we evaluated the results from the different combinations of pre-processing methods (Pipelines) and ML and DL models where we ascertained that LSTMs and HFAutoModel with a Bert embedding were the best models for the datasets we used. In general, DL and Autogluon models gave us better results than ML. For the pre-processing Pipelines, we were able to visualise that there is no one Pipeline fits all solution, each model had different Pipelines working better.

Lastly, we will discuss the conclusions we can take from this work along with a future work section, where we explore the possible next steps for this project.

Keywords: Deep Learning; Machine Learning; Text Mining; Sentiment Analysis

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence 1, 2, 4, 36, 42
CNN	Convolutional Neural Networks 13, 15, 41, 57
DL	Deep Learning vi, vii, 1, 3, 12, 13, 19, 20, 22, 28, 34, 40, 41, 42, 44, 56, 57, 59, 60, 62, 63
EDA	Exploratory Data Analysis 28, 35, 36, 44, 45, 61
FFNN	Feed Forward Neural Networks 13, 14
LSTM	Long Short Term Memory vi, vii, 15, 41, 57
ML	Machine Learning vi, vii, 1, 3, 4, 6, 7, 9, 10, 11, 12, 19, 20, 22, 24, 25, 26, 28, 34, 40, 42, 44, 56, 57, 59, 63
NLP	Natural Language Processing 2, 3, 15, 18, 20, 21, 22, 23, 36
NN	Neural Networks 1, 12, 15, 26, 41
RBM	Restricted Boltzmann Machines 13
RL	Reinforcement Learning 10, 11
RNN	Recurrent Neural Networks 13, 14, 15, 41, 57
SA	Sentiment Analysis vii, 1, 2, 3, 21, 22, 23, 28, 34, 35, 45, 61, 62, 63
SL	Supervised Learning 5, 6, 7, 9, 10, 11
TL	Transfer Learning 3, 19
TM	Text Mining 2, 3, 20, 21, 22, 32, 62

UL	Unsupervised Learning 7, 8, 9, 11, 16

Chapter 1

Introduction

The first chapter gives an introduction to the research conducted in this dissertation. The intention is to provide a sharper look into this dissertation's context, motivation and objectives.

1.1 Context & Motivation

SA (or Opinion Mining) is the process through which one can analyse segments of text in order to determine the sentiment behind it.

SA can be used for determining the market sentiment on any given topic, which makes it a powerful tool for researchers. SA can also be used for brand monitoring through social media and to identify customer intent.

The SA process typically involves the use of ML models trained with labelled examples of emotions in text. The ML models can then detect emotions on unlabelled inputs without the need for human intervention. ML allows computers to learn tasks without being expressly programmed to perform them.

DL can also be applied in sentiment analysis. DL is an ML technique that uses Neural Networks (NN) to extract features from the input to train a model to make predictions on a given set of data. NNs works by trying to identify patterns in data with close to no human interaction. DL and ML techniques have been very successful in performing sentiment analysis and are used daily by all the big brands.

SA models can focus on different types of sentiments. These sentiments range from the analysis of polarity (positivity, negativity and neutrality) through feelings/emotions (such as anger and happiness) to urgency/intention (is it urgent?, is it interesting?). Due to this versatility, sentiment analysis is a powerful and important technique in Artificial Intelligence (AI). However, despite the importance of this field and the numerous studies on it, major challenges in its use can still be found.

One of the most significant challenges it faces is the difference between context and polarity, and the addition of sarcasm and irony in the texts. In some use cases, these challenges are non-problematic, but in most cases involving feelings and emotions, finding solutions to these problems may help the models become more precise. Other common challenges in SA are how to deal with comparisons, emojis, the definition of neutral and the accuracy of the inputted data. Not less important is that most tools are built for the English language, and multilingual tools are few and mostly do not involve the Portuguese language.

The main goal of this dissertation is to develop Text Mining (TM) tools able to analyse and determine the sentiment behind segments of text in the Portuguese language. TM is the process of Natural Language Processing (NLP) that tries to understand and sort text to make it easier to manage. NLP is a sub-field of AI that concerns the interactions between computers and human (natural) languages, particularly helping computers understand, interpret and manipulate human (natural) languages.

This tool will prove to be valuable in several different use cases.

1.2 Objectives

The main goal of this work is to develop a computational tool that allows one to perform SA for texts written in the Portuguese language. The tool will be based on training and validating machine/ deep learning models for SA, but also taking into account methods and algorithms from NLP and TM for data pre-processing.

This dissertation was done within the framework of Omnium AI, with the aim of creating a tool the company can use for sentiment analysis in Portuguese Texts.

The work will approach several scientific/ technical objectives starting with an in-depth review of the relevant literature regarding SA and its applications, focusing on SA applied to Portuguese texts.

As the second objective, we will collect datasets for SA written in the Portuguese language from different sources, focusing on social networks, more precisely from Twitter.

As a third objective, we will develop and compare various data pre-processing methods and workflows and assess the impact on the quality of predictive models.

As a fourth objective, we will train and evaluate the performance of different machine/deep learning models for SA in different datasets in the Portuguese language.

As a fifth and final objective, we will develop a software framework that encompasses data collection, data pre-processing, model training and validation for SA, addressing the automation of the model optimisation process.

1.3 Document Structure

Chapter 1 defines the main objectives of the endeavoured work and presents an overview of how it is partitioned.

Chapter 2 discusses the State-of-the-Art which will be composed by the current landscape on SA, with a special focus on the Portuguese language. The chapter also addresses more general subject matters like ML, DL, Transfer Learning (TL) and NLP with emphasis on ML supervised and unsupervised models but also on DL models. Additionally, the chapter presents the most common o NLP and TM techniques and how they may help get better results and efficiency.

Chapter 3 overviews the problem we are tackling with this dissertation and the framework in which we are going to be working. This framework includes an explanation of Omnium AI's architecture and guidelines and the technologies that are going to be used.

Chapter 4 addresses the development of the tools we need to achieve our goal. It describes how we got our data, the pre-processing methods we created for our package and the models we developed.

Chapter 5 denotes the results of applying our tools to a specific use case. It focuses on the gathering of data, its pre-processing and the validation of the models we trained with our data.

Lastly, chapter 6 presents us with the conclusions from our dissertation and the future work that can be done to improve this work.

Chapter 2

State of the Art

2.1 Machine Learning

Nowadays, ML is one of the biggest buzzwords in Computer Science, and deservingly so. It is currently one of the most exciting sub-fields of AI, which is proven by the great results being presented using AI methods. The term "Machine Learning"was coined by Arthur Samuel in 1959. He defined it as the *"Field of study that gives computers the capability to learn without being explicitly programmed"*.

In such a manner, we can define ML as a data analytics technique in which a computer is taught what normally comes naturally to humans. ML uses computational methods to learn information directly from data. These algorithms can improve their performance as the number of samples available for learning increases. ML uses a couple of techniques for learning. The three most common are unsupervised, supervised, and reinforcement learning [1].

There are also two types of learning [2], the first one is lazy learning, where one store the training data and wait until the test data appears. The classification is done using the most similar data in the training set. This type of learning is slower than the second type, which is eager learning. In eager learning, one constructs a model using the training data, before getting the test data to make the predictions. This takes much time in training, but it is much faster when predicting.

In the last years, ML has been gaining more and more applications due to the many benefits it provides. Even though ML is very powerful, it too has its limitations, and we are going to check some of these in a section further ahead.

2.1.1 Supervised Learning

Supervised Learning (SL) is one of the main paradigms for Machine Learning. SL can be defined as the process that makes an algorithm map an input into an output. This can be achieved by using labelled datasets. If the mapping in the datasets is correct, the algorithm will successfully learn. SL will help us predict unseen future data. SL can be simply understood by looking at the figure (Figure 1).

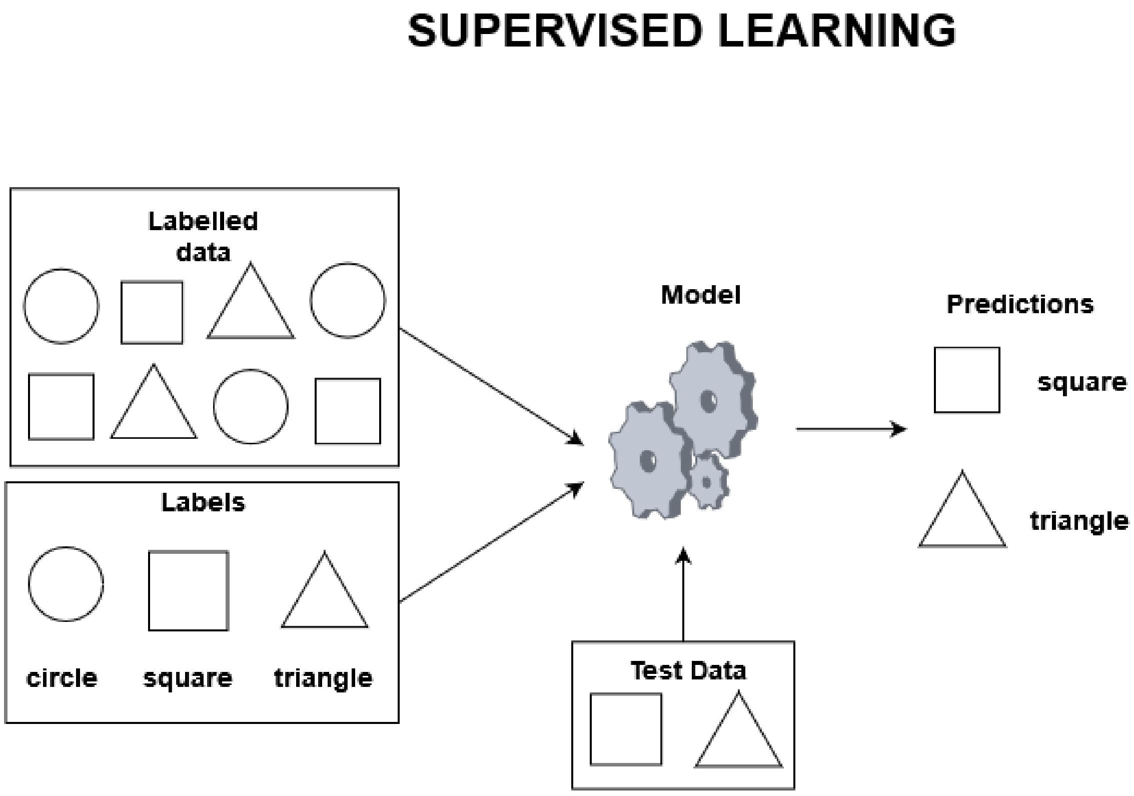


Figure 1: Supervised Learning Architecture, adapted from [3]

As one can conclude from the figure (Figure 1), SL needs a set of data and their labels. The data is used to create a model. The model aims to receive unlabelled data and predict its labels.

SL can be broadly divided into two types of problems: classification and regression[1].

The first type, Classification, occurs when the output variable is categorical (outputs such as yes-no, male-female, etc.). In general, classification is used to identify labels or groups. Classification can be categorised into two groups, the first being binary classification where the input variables are categorised into two labels. The second category is the multiclass (multinomial) classification where the input data is segregated into three or more groups.

There are several algorithms used in classification. One of the most popular classes is Bayesian algorithms (like Naive Bayes, and Bayesian Networks, among others).

Naive Bayes is an algorithm that generates a probability table from a classification technique. It is commonly used for statistics and machine learning. This algorithm rationale stands from Thomas Bayes' studies where he applies the Bayesian Theorem with a strong (naive) independence assumption between the features of the data. Naive Bayes are highly scalable and normally use Gaussian distribution (also known as Normal Distribution) to calculate the mean and standard deviation of the data. There are several different versions of Naive Bayes. In most cases, they follow the same rationale but use different distribution functions. For instance, Multinomial Naive Bayes is based on the Bayesian Theorem and it's used for different Natural Processing Language and Machine Learning applications. This algorithm differs from Gaussian Naive Bayes in the manner it calculates the distribution, by using multinomial distribution.

Random forest is another classification (and regression) algorithm which combines the output of multiple decision trees to reach a single result. Random forest algorithms have three main hyperparameters: node size, the number of trees, and the number of features sampled. The random forest algorithm is composed of a collection of trees, each tree consists of a data sample drawn from a training set with replacement. Then through feature bagging, it added more diversity to the dataset and reduced the correlation among decision trees, and added more randomness to the trees. In the end, for a classification problem, the class that was output more times by the trees in the predicted class. Random Forests present us with several benefits that include reduced risk of overfitting due to the number of robust trees being calculated, and provide flexibility because it can be used in both regression and classification problems. There are some setbacks such as its complexity, its time-consuming process and the requirement of more resources than other ML alternatives.

Algorithms that calculate Nearest Neighbours (algorithms like KNN) to classify according to similarities between neighbours are also extensively used. As previously stated, KNNs is an algorithm that calculates the nearest neighbours for their classification. It works by calculating the distance between unknown data points and all the training examples. Then it searches for the k (a predefined value) observations in the training data nearest to the measurements of the unknown data point and calculates the distance between the unknown data point and the training data. In the end, the training data which has the smallest value will be declared as the nearest neighbour and used for the classification of the unknown data point.

In this type of algorithm, the choice of k, and the metrics used to calculate the distances can have major impacts on the final results. This type of algorithm can be very computationally expensive compared to Bayesian algorithms, but it can be a simple and easy way to predict labels for small datasets.

Decision trees are another algorithm for classification that presents us with great results. Several other types of algorithms present good results for classification, like Support Vector Machines (SVMs) and discriminate analysis.

The second type of problem in SL is Regression [4]. Regression is a technique that predicts continuous or real values. We can have three types of regression; linear, multiple and polynomial regression. In linear regression, we have only one independent variable used to predict the output. In multiple regression, we have more than one independent variable used to predict the output, and in polynomial regression, the relationship between the dependent and independent variables follows a polynomial function.

Ensemble Methods (like Bagging, Boosting and Stacking) are also used to reduce variance and thereby increase the accuracy of predictions. Ensemble methods work by combining different models to create better predictive models, and they can be used in both classification and regression algorithms. Bagging works by combining a bootstrapping step followed by an aggregation step to increase accuracy by reducing variance. The reduction of variance comes from the elimination of overfitting, helping models to reach higher accuracies. With the Bootstrapping step, we derive samples from the whole population using a replacement procedure, which helps to randomise the selection procedure. Then the base learning algorithm is run on the samples. With the aggregation step, we incorporate all possible outcomes of the predictions and compute an outcome. Without this step, the predictions will not be accurate because we will not consider all the outcomes.

Boosting works by learning from the last predictor mistakes to make better predictions in the future. This technique combines several weak base learners to form one strong learner, improving the predictability of models. It works by arranging weak learners in a sequence, such that weak learners learn from the next learner in the sequence to create a better predictive model.

Stacking works by allowing a training algorithm to ensemble other similar learning algorithm predictions.

SL is important due to the many benefits it provides us. It lets us be specific about labels and classes we use in the training data, helping us define perfect boundaries between classes. It is easy to understand the process in comparison to, for example, Unsupervised Learning (UL). Even though SL gives us so many benefits, it also provides limitations. SL is prone to overfitting algorithms. Overfitting happens when a model fits with its training data, making it perform really good with this data but not achieving accurate results with new data inputs. There are several paths to avoid overfitting, one of the most common is the utilisation of the aforementioned ensemble methods (bagging, boosting and stacking). Another limitation is if the training data is not good (i.e incomplete data or wrongly labelled data), the algorithms will give wrong predictions, and the computation time may be very large (even more significant for big training datasets). We also have to consider pre-processing in supervised learning, which can be challenging.

2.1.2 Unsupervised Learning

Unsupervised Learning is another technique for ML, and it differs from SL because we do not need to label data. That means that UL uses unstructured data to learn structures and patterns in the data.

We can understand how UL works by looking at the following figure (Figure 2).

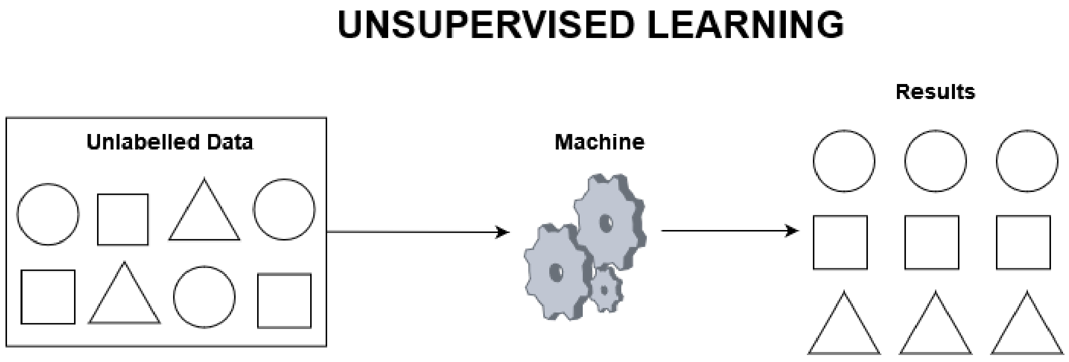


Figure 2: Unsupervised Learning Architecture, adapted from [3]

As we can see in the figure (Figure 2), UL receives an unlabelled set of data and outputs the data organised according to the similarity between them. UL can be broadly divided into three types[1]: Clustering, Association Rules and Dimensionality Reduction.

Clustering

In Clustering [5], we have a technique that tries to divide the data into a number of groups where data points in the same group are more similar between themselves than data points outside their group. It creates a cluster of different data points with a lower inner variance There are several methods to cluster data. The four most common are density-based, hierarchical-based, grid-based and partitioning methods. In Density-Based methods, we consider the clusters as the dense region that has some similarities between points in that region and differences from another dense region of the space. The most common density-based algorithms are DBSCAN [6] and OPTICS [7]. In Hierarchical based methods, the clusters formed are in a tree-type structure based on a hierarchy, which can be performed by a bottom-up approach (i.e. agglomerative approach) or a top-down approach (i.e. divisive approach). The most common hierarchal-based algorithms are CURE [8] and BIRCH [9]. For Partitioning methods, we partition the objects into k clusters, and each partition forms one cluster. The most notable partitioning algorithms are K-Means [10] (and its multiple variants, like k-Mode [11]) and CLARANS [12]. The last methods for Clustering are the Grid-Based methods. In Grid-Based methods, we formulate the data space into a finite grid-like structure of cells. The operations to the cluster data points are performed on this grid. The most notable grid-based algorithms are STING [13] and CLIQUE [14].

Association Rules

In Association Rules [15], we try to find relationships between the data in a given dataset. Association Rules are descriptive models and not predictive ones, which means they do not predict labels, but instead show us interesting relationships between data points.

Association Rules algorithms follow a two-step approach, where we first generate the frequent itemset and then generate the rules. The first step can be computationally expensive. Association Rules can be divided into three types of algorithms. These are Apriori [16], Eclat [17] and FP-Growth [18].The Apriori algorithm uses frequent datasets to generate association rules and uses a breadth-first search and Hash Trees to calculate the itemset efficiently. The Eclat (Equivalence Class Transformation) algorithm is a method that uses a depth-first search technique to find the frequent itemsets in a dataset. It is more efficient than Apriori Algorithm. FP-Growth (Frequent Pattern) represents the dataset as a tree called a frequent pattern tree. The purpose of this frequent tree is to extract the most frequent patterns in the dataset. FP-Growth is another improved version of the Apriori Algorithm.

Dimensionality Reduction

In Dimensionality Reduction [4], we map the data points in a lower dimension space. A simple way to see this is to try and map a 2D space in a line. The following figure (Figure 3) depicts a reduction from a 3D space to a 2D space.

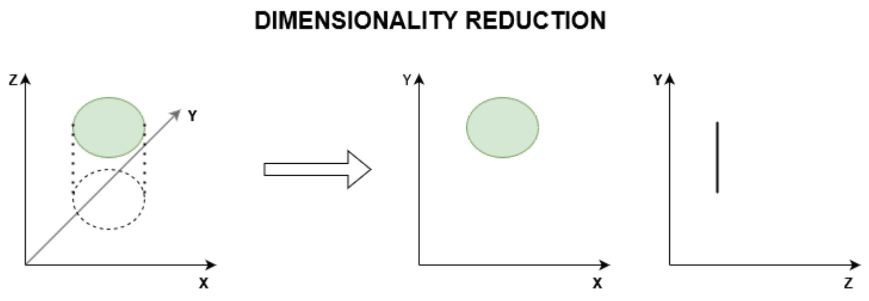


Figure 3: Dimensionality Reduction from 3D to 2D, adapted from [19]

Dimensionality reduction works by reducing the number of features in a given set. There are two components to dimensionality reduction, we have a feature selection component that tries to find a smaller subset of the original set of variables (i.e. features). This subset is then usually used to model the problem. The second component of dimensionality reduction is feature extraction where we reduce the data in a high dimensional space to a lower dimension space.

There are several methods for reducing the dimension we are working with, but the two most notable are principal component analysis (PCA) [20] and latent Dirichlet allocation (LDA) [21]. PCA works by transforming the features of a dataset into a new set of features called principal components. This transformation aims to compress the information present in the dataset into fewer features. LDA achieves dimensionality reduction by using a Dirichlet distribution.

As we have seen, UL is very powerful and lets us see what human minds may not even visualise, it can dig hidden patterns. It can be less complex in comparison to the other types of techniques for ML. It is easier to start because we do not need labelled data. UL has its disadvantages, such as being costlier than SL since human intervention may be needed to understand the patterns and correlate them with the specific use case being studied. The results are not always useful since there is no label or output to measure their usefulness. The results are often less accurate (in terms of accuracy in predictions) than the other techniques.

2.1.3 Reinforcement Learning

The last technique for ML we are going to talk about is Reinforcement Learning (RL) [4]. It works by taking suitable action to maximise reward in a particular situation. RL is different from SL because the training data has the correct output in SL, so the model is trained with the right answer. In contrast, there is no answer in RL, the reinforcement agent decides what to do at each step to perform the given task, maximising a reward function. Because it does not have a training set, it is bound to learn from its experience. The training is based on the input (the initial state from which the agent starts). The agent will return an action, and the user decides to reward or punish the model based on the output. The agent keeps learning, and the best solution is decided based on the maximum reward.

There are two types of reinforcement. One is positive reinforcement, where a positive effect will be seen in an event. The second type of reinforcement is negative reinforcement, which works as the exact opposite of positive reinforcement. We can easily understand RL with the figure below (Figure 4).

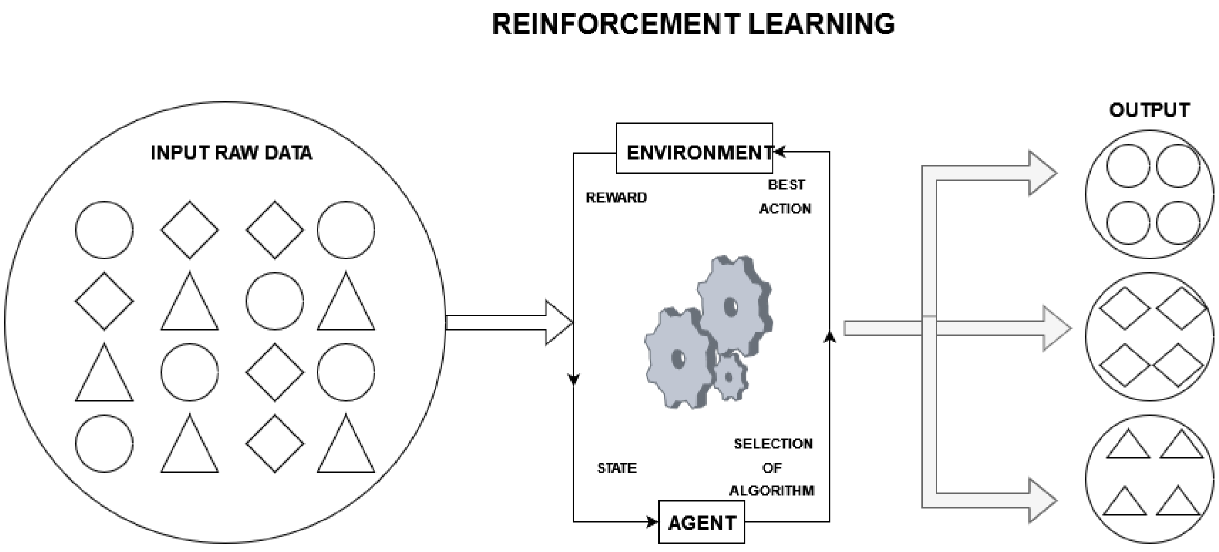


Figure 4: Reinforcement Learning Architecture, adapted from [22]

As we can see, we input raw data to the environment then we have a cycle where we are learning by experience through the use of the agent. The agent will adapt its learning in accordance with the rewards it gets from the environment.

There are several algorithms and approaches for RL. The most notable ones are Markov Decision Processes (MPDs) [23], SARSA [24], Q-learning [25].

Markov Decision Processes is the root concept for the process of learning from the current state and actions. This process has 4 important areas: states, actions, transitions and a reward function. It works by taking the current state information to make a decision. The reward function determines what action to take in order to make the transition to the next state. This methodology is the base for most reinforcement learning algorithms. Q-learning uses this methodology using a Quality that depicts the quality of the action chosen. Q-learning is an off-policy method, meaning it is a greedy learner because it learns from random actions. SARSA stands for state-Action-Reward-State-Action and is similar to Q-learning with the main difference being it's an on-policy method, meaning it learns from the current state and actions.

RL is a cutting-edge technology with a very powerful set of applications with no need for a large amount of labelled data, it is an innovative technique that is imitating whoever provided the data. It is biased resistant because it does not have a bias from the beginning as SL has from its labelled data which a lot of times may be biased (from the labelling methodology). RL is adaptable and goal-oriented, which are another two advantages, but it has a couple of disadvantages like the need for a large amount of data (from experience). As the complexity increases, the algorithms need more and more data to be acquired from interaction with the environment. The results of RL models depend on the agent's exploration of the environment, bringing limitations to the model.

2.1.4 Applications

There are many applications for ML, which range from several areas. Companies across all industries use SL to address issues like customer churn detection [26], customer lifetime worth assessment [27], personalising reviews for goods, human resources distribution, sales forecasting [28], supply and demand analysis [29], identifying fraud [30], predicting repair of equipment. It is to notice that given a well-labelled dataset, we can predict pretty much everything, using supervised learning methods, provided there is a statistical relationship between inputs and outputs.

UL is used by companies across all industries to address issues like anomaly and similarity detection [31], products and customer segmentation [32], recommendation systems [33], and even labelling of datasets to be used in supervised methods.

RL is also being used by companies across all industries. For instance, it is being used in the gaming industry with much success for some time now. It is also used to test reinforcement algorithms by training them to learn how to play certain games such as chess, go [34] and Othello [35]. RL is used in the financial industry to generate more return on investment, reduce cost, and improve customer experience. It can also improve execution while approving loans, measuring risk factors, and managing investments, among other use cases in the financial industry and financial markets. RL can also be used in the healthcare industry; two big examples are the healthcare domain Quotient Health [36] and the application KenSci [37]. Quotient Health uses reinforcement learning to reduce the expenses of electronic medical records to improve healthcare systems with lower costs. KenSci tries to help medical practitioners and patients intervene in earlier stages of diseases by using reinforcement learning to predetermine ailments and treatments. A lot more use cases use RL like the marketing industry, image processing and even the robotics industry.

2.1.5 Limitations and benefits

After explaining ML, in a general way, two questions remain: *Why is Machine Learning important? What are its Limitations?*

ML is important because it can easily identify trends and patterns, it (mostly) does not need human interaction, it can handle all types of data, and consequently, impact many industries. Machine Learning keeps learning and thereby keeps improving with time. ML is energy and time efficient and can work in a big range of applications which can help companies gain money with little investment. However, ML has its limitations too. Limitations such as being prone to errors and being very expensive for more complex use cases. From the data acquisition costs to the time and resources to create models, and to the people needed to create tools and interpret results makes it almost impossible for smaller businesses to use it. But with the growth of the area inevitably it will be easier and cheaper for everybody.

A limitation that is sometimes overlooked is the automation itself, ML is not the right answer for every use case because automation itself may take a bit of personalising of methods out of the equation. Another limitation is the lack of good data. As ML is a relatively recent field, a lot of areas need to be explored and the biggest problem right now is having good data, which is an essential element in machine learning. Another big limitation of ML models is overfitting, which occurs when a model corresponds closely to the training data. This makes the model good at predicting data similar to its training data but if the test data is not similar to the training data the model will not be able to have good predictions.

2.2 Deep Learning

Deep Learning is a Machine Learning technique based on NN [38]. It works by using multiple layers to extract features from the input, thus learning the best representation from the data.

We can better understand how DL works with the figure below (Figure 5). As we can see, we have a set of input values that feed the input layer. The input is then passed through several layers (in this case, 2 hidden layers) where information is extracted and given to the user.

In this section we are going to discuss a bit about NN, going deep into some of the most common types of networks. We are going to deliberate on the best way to train models. At the end of the section, we will debate the main differences between ML and DL and the limitations and benefits of DL.

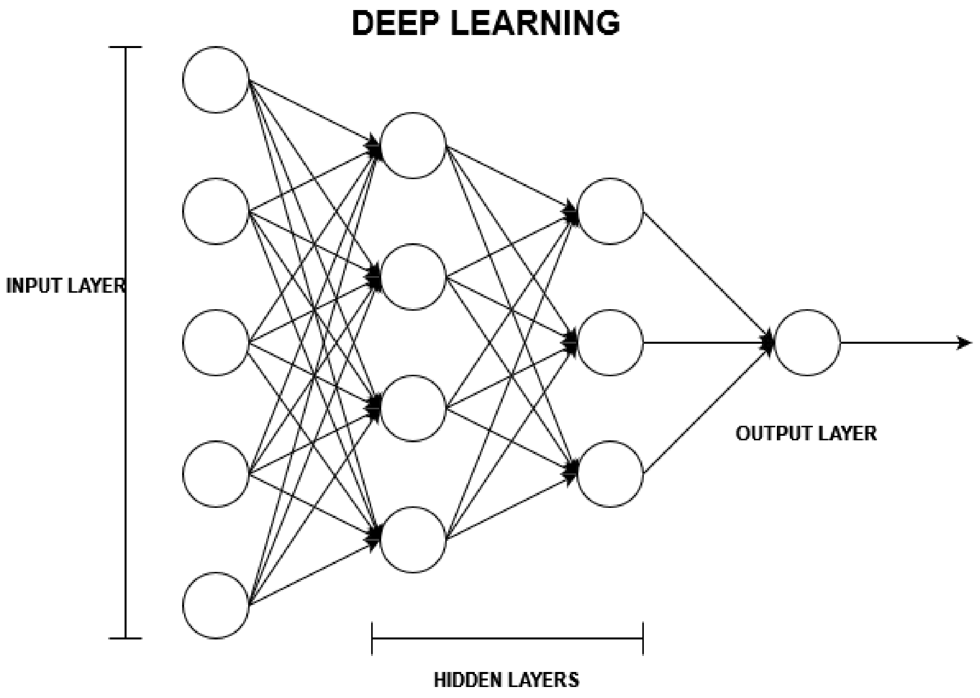


Figure 5: Deep Learning Architecture, adapted from [39]

2.2.1 Artificial Neural Networks

Artificial Neural Networks, normally called Neural Networks, are computing systems inspired by biological brains. They are based on a collection of connected nodes (artificial neurons). Each connection (called edges), like biological brains (with synapses), can transmit a signal to other nodes [40].

An artificial neuron receives a signal, processes it and signals the next neuron (connected with it). The 'signal' is a number and the output of each neuron is typically computed by applying a non-linear activation function to the sum of its inputs multiplied by the weights of connecting edges. Indeed, edges connecting neurons have weights that are adjusted through a learning process. Each weight increases or decreases the strength of the signal at their respective connection.

In feed-forward networks, neurons are typically aggregated into different layers, with each layer being able to perform different transformations on their inputs. Signals go from the first layer to the last layer.

Artificial Networks can implement the different machine learning paradigms (Supervised, Unsupervised, Reinforcement).

DL has a few types of networks and technologies with the main ones being Feed Forward Neural Networks (FFNN), Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), Restricted Boltzmann Machines (RBM), Autoencoders, Embeddings and Transformers.

FeedForward Neural Network

A Feed-Forward Neural Network is a network that ensures that the nodes do not form a cycle [41]. All the neurons are organised in interconnected layers. In FFNNs, nodes in the same layer cannot be interconnected, making it impossible to have back loops. To minimise prediction errors, the backpropagation algorithm can be used to update weight values. These types of networks have several applications like data compression [42], pattern recognition [43], computer vision [44], sonar target recognition [45], speech recognition [46], handwritten character recognition [47].

Recurrent Neural Network

Recurrent Neural Networks [48] are an evolution of Feed-Forward Neural Networks. In RNNs, each of the neurons present in the hidden layers receives an input with a specific delay in time. RNNs work by saving the output of a layer and feeding it back to the input to predict the output. With the figure below (Figure 6), we can easily visualise this.

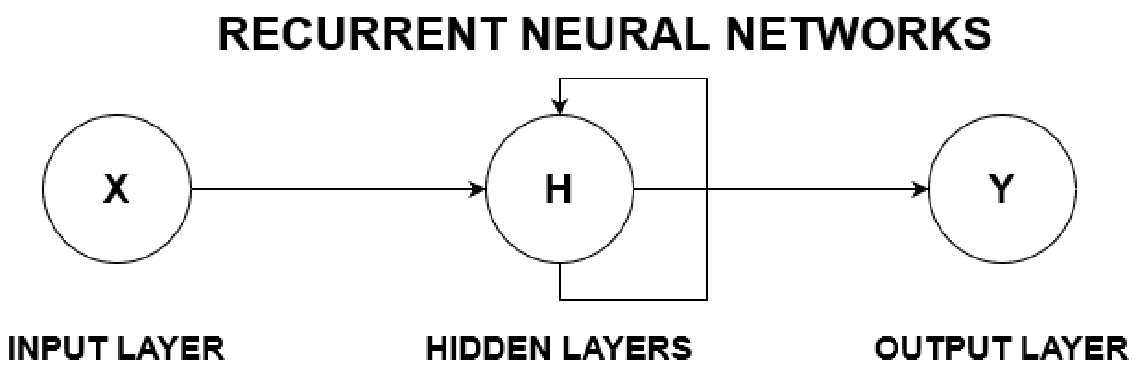


Figure 6: Recurrent Neural Network, adapted from [49]

As we can see in the figure (Figure 6), RNNs use a loop to iterate to the hidden layers, giving them the capability to remember previous states. This loop gives the model a memorisation capability that lets it predict outputs with the knowledge gained in previous states. There are four types of architectures in RNNs. These are One-to-One, One-to-Many, Many-to-One and Many-to-Many. The names relate to the number of inputs and outputs, and the format of the names of these types denotes InputNumber-to-OutputNumber. In the Many-to-Many architecture, we have a set of three matrices. The first is between the input and hidden layer, the second is between hidden layers and the third is between the hidden layer and the output layer. The model will learn by using these matrices and a couple of bias vectors. These parameters often use the back-propagation through time algorithm [50], where after a forward pass, we perform a backward pass adjusting the model's parameters. This helps to repeatedly adjust parameters so that it can minimise the cost function.

RNNs were created to tackle two main issues of feed-forward networks. These issues are the incapability of those networks in handling sequential data and memorising previous inputs.

The most used RNNs are LSTMs (Long Short Term Memory) [51] and GRU (Gated Recurrent Unit) [52]. LSTM is a RNN that was created to deal with the two main problems that RNN deal with. LSTM prevents the output of the model from either decaying or exploding as it cycles through feedback loops. These feedback loops allow the network to be better at pattern recognition.

These networks are used for different applications like machine translation, robot control, time series prediction [51], speech recognition, speech synthesis [53], sequence modelling [52], music composition [54].

Convolutional Neural Network

Convolutional Neural Networks [55] are NN that differentiate themselves from other NNs by having superior performance in the classification of visual content, recognition of objects within its scenery (for example, street signs), and group the recognised objects into clusters (e.g. facial recognition on a pool of faces). It can accomplish this by having an architecture composed of three main layers (convolutional layer, pooling layer and fully-connected layer). The convolutional layer is the first layer of a CNN, it is followed by other layers (including more convolutional layers and pooling layers) and it has a fully connected layer as its last layer. With each layer, CNNs can identify more complex portions of what we are analysing, in the case of an image the first layers try to identify edges, colours and other simple features. As the process evolves, layers start to recognise the shapes of objects until they finally identify the object in the image. The convolutional layer is the main building block of the NNs. It needs to have the input data, a filter and a feature map. The filter is a feature detector that will search the input for the feature, this process is called convolution. The pooling layer, also known as downsampling reduces the number of parameters in the input (dimensionality reduction). Similarly to the convolutional layer, it has a filter that will search the input for a feature, with the difference that this filter doesn't have weights. It works by using an aggregation function to populate the output array using the values in the receptive field. There are two types of pooling: max pooling where the filter sweeps the field choosing the maximum valued pixel to send to the output, average pooling that calculates the average value of the receptive field. Even though information is lost by reducing the number of parameters, this layer helps to reduce complexity, improving efficiency and reducing the probability of overfitting. The fully connected layer, also known as the dense layer, performs the task of classification using the information extracted in previous layers. Being a fully connected layer helps by connecting every node to the output layer.

These NNs have a lot of applications with the main ones being able to identify faces, street signs[56], tumours [57]. It can also recognise images, analyse videos and detect anomalies.

Other applications of these networks can be NLP, Drug Discovery, Time Series Forecasting [58], among others.

Restricted Boltzmann Machine

Restricted Boltzmann Machines are a variant of Neural Networks where the neurons of the input layer and the hidden layers encompass symmetric connections amid them. Nevertheless, there is no internal association within their layers, unlike traditional Boltzmann machines. These restrictions in Boltzmann Machines help the model to train more efficiently [59].

Applications for these networks range from filtering, feature learning, classification, risk detection, business and economic analysis.

Autoencoders

An autoencoder is a UL algorithm where the number of hidden cells is smaller than that of the input cells. However, the number of input cells is the same as the number of output cells. These networks are trained to display the output similar to the inputs to force them to find common patterns and generalise the data. The autoencoder networks are mainly used to get smaller representations of the input. They work by following two steps. An Encoder step that converts the input data into lower dimensions and a Decoder step that reconstructs the compressed data from that latent representation.

The main use cases for autoencoders are classification, clustering and feature compression.

Embeddings

An embedding [60] is a technique that can represent high dimensional data in low dimensional representations. A good embedding will not capture all the information in the original data but instead capture enough to solve the problem being solved. Embeddings work by using the same idea as dimensionality reduction. So an embedding will reduce the dimension of the original input while maintaining the essential data to solve the problem. Embeddings are often used with distance matrices to solve real-world problems.

Embeddings have several applications, from word embeddings (like word2vec [61]) to social media [62]. There are many well-known embeddings like PCA [20] and Multidimensional scaling (MDS) [63].

Transformers

In 2017, Vaswani et al. [64] proposed a new architecture that could maintain the attention mechanism while processing sequences in parallel. By doing this we could process all words together relatively to a word-by-word basis. With this architecture, parallelism became real. The use of parallelism gives us a solution that is computationally inexpensive in comparison to sequential solutions.

Transformers are a classic sequence-to-sequence model which architecture as two interdependent segments: an encoder segment and a decoder segment.

This process can be simply understood by the figure below (Figure 7).

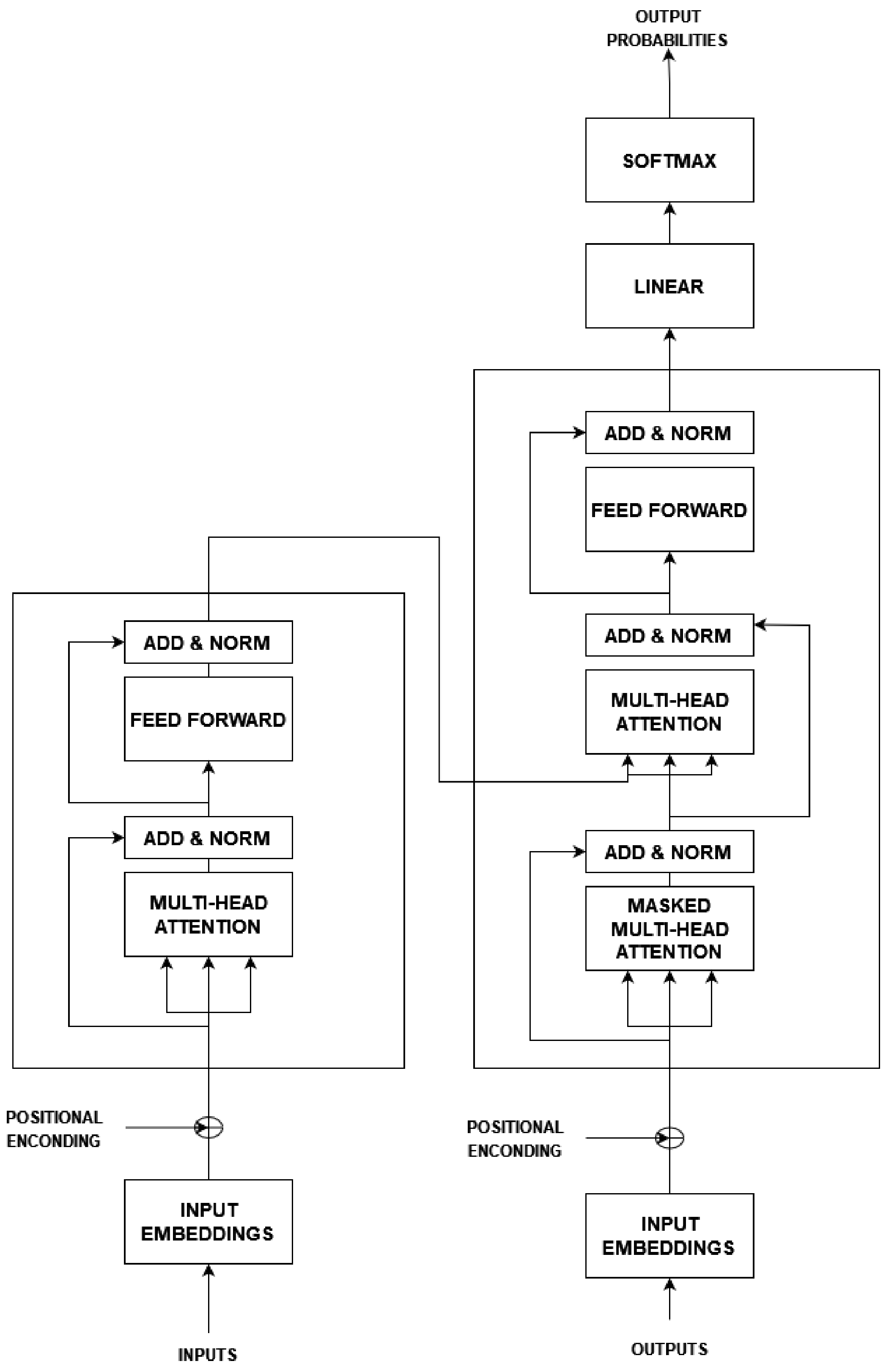


Figure 7: Transformers, adapted from [64]

The encoder segment takes the inputs from the source language and generates an embedding for them encoding the positions. Then it computes where each word has to attend in the multi-context setting. Finally, it outputs an intermediary representation.

The decoder segment takes inputs from the target language and generates an embedding for them encoding the positions. Then it computes where each word has to attend to, with what was already produced. Afterwards, combines it with the encoder output with what it has already produced. The outcome is a prediction for the next token (using softmax and hence argmax class prediction, where each token or word is a class).

In the encoder segment, we have the ability to transform inputs into an intermediary language. This segment is composed of a couple of individual components (input embeddings, positional encodings, the actual encoder segment). The input embedding converts tokenized inputs into a vector so that they can be used. Positional encodings change the vectors created in the input embeddings and add positional information to these vectors. The actual encoder segment can be divided into two sub-segments. The first sub-segment is a multi-head attention segment that performs a multi-head self-attention adding the residual connection and performing the normalisation of the layer. The second sub-segment is the feed-forward segment that generates the encoder output for each token. These sub-segments can be repeated N times ([64] chose N = 6).

In the decoder segment, we can convert the intermediary into predictions for the output tokens. This segment has a couple of individual components (output embeddings, positional encodings and the actual decoder). The output embedding converts tokenized outputs into vector format (like the encoder with the input embeddings). Positional encodings add positional information to the vectors created by the output embeddings. The actual decoder can be divided into three subsegments. The first is the masked multi-head attention segment that performs a multi-head self-attention on the outputs but does it in a masked way so that the positions only rely on the past. The second is the multi-head attention segment that performs a multi-head self-attention on a combination of the outputs and the encoded inputs. The purpose of this step is for the model to learn to correlate encoded inputs with outputs. The last segment is the feed-forward segment that processes the tokens individually.

There are several transformers implemented like Google's BERT that revolutionised NLP [65]. BERT also known as Bidirectional Encoder Representations from Transformers, is pre-trained on a large corpus (including the entire Wikipedia). Being bidirectional means it learns from both the right and the left of a token during the training phase which makes it easier to learn the context of a phrase.

2.2.2 Training Models

There are two main ways to train and create deep learning models. The first way is by training from scratch, where we gather a large amount of data and design a network architecture that can learn the features we need. The other way to train is Transfer Learning [66], where we fine-tune a pre-trained model. We start with an existing network and feed it new data containing previously unknown classes.

After making some changes to the network, we can perform new tasks.

TL needs much less data, which massively reduces computation times.

2.2.3 Machine Learning vs Deep Learning

We have explored how DL works, but why would we use it instead of traditional ML techniques? There are several differences between DL and ML [67]. In this section, we are going to explore some of them.

The first difference is the amount of data; DL works with a larger amount of data while ML works with smaller datasets to improve accuracy. This difference can be a helpful indicator of what to use depending on the amount of existing data. A second difference is that DL commonly relies on high-end machines while ML can work on lower-end machines. DL needs higher-end machines due to the huge amount of operations needed (and it gets even worse because we normally get more data than in ML). The bigger amount of computational operations means it might be too much for CPUs to run it (in a reasonable amount of time) making DL users have GPUs (computing units that have a higher capability to compute operations faster than CPUs due to parallelism and a more specialised set of operations). Another difference is time; complex DL models normally take longer to train compared to traditional ML algorithms. In contrast, DL normally takes less time to test.

2.2.4 Limitations and benefits

The last thing we are going to talk about regarding DL is its limitations and benefits. We will start by stating the limitations. The most significant limitation is learning through observation, which means that models only know whatever is on the data they train. If there is only a small amount of data, the model will learn very little information, resulting in an inaccurate model. The second biggest limitation is bias. Data can contain bias, which has been a vexing problem in the area. Models recreate the bias in the data, consequently resulting in biased models. One big example is biased facial recognition software that uses racial bias in its recognition. This has been happening everywhere, with a good example being the UK's passport checking system that was knowingly being used even though the governing bodies knew the inaccuracy of correctly identifying faces with darker skin tones [68].

There are other limitations like the learning rate. If too high, it gives a model that converges too quickly, producing a less than optimal solution. If it is too low the process may get stuck, and it will be even harder to reach a solution. Another limitation is the hardware requirements that can be prohibitive due to the high cost of high-end machines. There are many more limitations like a large amount of data, which may be a problem in terms of storage space and processing power. It can also be a problem in terms of overfitting and bias.

We will now talk about the benefits of DL. Let us start with the most obvious, in terms of performance it beats the other types of ML. Another benefit is the reduction of the need to do feature extraction. It can also eliminate costs by being able to identify defects easily that otherwise would be difficult to detect.

Another benefit from DL is it provides great results with unstructured data in comparison to traditional ML models. The last benefit we are going to discuss is scalability, in DL we have a more scalable solution to our problem due to the ease of working with bigger amounts of data in comparison to other solutions.

2.3 Text Mining

With the advances in technology, more and more data are available. With most of these data being unstructured text, there was the need to develop techniques to extract knowledge from these data. Consequently, the area of Text Mining flourished.

Hence, the area of Text Mining can be defined as the process of transforming unstructured text into structured data for more straightforward analysis. TM accomplishes that by using NLP to allow machines to understand human language.

In general, TM can be divided into three areas: Information Retrieval (IR), Information Extraction (IE) and NLP. In the following subsections, we will talk about these areas of TM in more detail, starting with the first area, Information Retrieval.

2.3.1 Information Retrieval

Information Retrieval [69] can be defined as the process of retrieving useful data from the unstructured data stored. Common techniques used for this area are summarization, compound term processing, and cross-lingual search. IR has many applications, mainly information filtering (like we find in recommender systems) and search engines.

2.3.2 Information Extraction

Information extraction as the name tells us, this area tackles the extraction of data from texts [70]. It can be defined as the process that deals with the automatic extraction of structured data from unstructured data sources. Typically, the process of IE follows a couple of steps. It starts with a pre-processing of the text, which is accomplished by using techniques like tokenization and sentence splitting. It is commonly followed by a step that tries to find and classify concepts where we detect and classify mentions of people, things, locations, events, and other pre-specified concepts. After detecting and classifying concepts we try and connect them by identifying relationships between the extracted concepts. In the end, the data is unified, meaning the data is put into a standard form where noise is eliminated with techniques like eliminating duplicate data. The process of IE can be entirely automated.

The most common applications for IE are information extraction in digital libraries, emails, personal profiles in social networks among many others.

2.3.3 Natural Language Processing

Natural Language Processing [71] is a component of Artificial Intelligence to get computer software applications to understand human language. The development of NLP applications can be hard because computers don't, generally, receive information in human speaking form. They normally receive it in a programming language that is a structured, simple way to transmit information.

Human speech is usually not well structured, so it can depend on many complex variables like slang, social context, and regional dialects, which is a complex situation we face while creating NLP applications.

There are many useful methods for the NLP area, with the main ones being Named Entity Recognition, Summarization, Topic Modelling, Text Classification, Keyword Extraction, Lemmatization, Stemming among many others.

With NLP being such as powerful area, many use cases exist, with the main ones being Autocorrect and Spell-check, Text Classification, Sentiment Analysis, Question Answering, Caption Generation, among many others.

NLP is a term that can, sometimes, be used interchangeably with TM, especially when speaking of NLP for text data.

2.3.4 Limitations and benefits

The biggest limitation of TM has nothing to do with technology but with copyright legislation [72]. The right to copy, digitise, and then to text mine is severely curtailed by the necessary restrictions placed on many texts to preserve the rights of copyright holders.

The second most significant limitation is the potential to gather garbled or false results due to poorly formed questions being asked of data or the nature of the text(s) under study.

Even though TM has its limitations, its benefits outweigh them a lot. The ability to access more facts and uncover relationships are probably the two main benefits for TM that make it a powerful tool for virtually every area.

2.4 Sentiment Analysis

We are living in an Information Age, where an abundance of data is created both by humans and machines. With data being created at an unprecedented rate, it is nearly impossible to gain insights into such data manually. The software comes to the rescue by letting us acquire insights into data by assessing it purely computationally.

One of the biggest areas explored is Sentiment analysis. Sentiment Analysis refers to the process in which computers try to find patterns to infer the emotion of a given piece of information.

SA works by following a simple life cycle that comes down to retrieving information, analysing it to get the emotions in the information gathered and then typically classifying the texts based on these emotions.

SA can be divided into four main types[73]: Fine-Grained Sentiment Analysis, Emotion Detection, Aspect Based Sentiment Analysis and Intent-Based Sentiment Analysis, which will be explored in the following sections.

In general, SA follows 3 simple steps which encapsulate its essence. The most important step, considered to be the first one, is: we have to understand the data. It is used, for example, with assessing the engagement with customers and delineate the causes so that we can improve their service later.

Nevertheless, analysing data is often difficult to perform, since data can be unstructured, opinionated and in some cases incomplete. These data have to be cleaned, to mature into usable data. In most cases, these cleanse must focus on deducing the polarity of the information, while clearing pieces that might lead to ambiguity.

At the end of the cleanse, we want to have the information good enough to deduce the polarity easily. We can use techniques like NLP [74], TM [75] to help prepare and analyse data for SA. Other areas like ML [76] , DL [77] and others are also used for SA.

2.4.1 Fine-Grained Sentiment Analysis

The first type we are going to talk about is the Fine-Grained Sentiment Analysis [78] where we try to capture the polarity of the inputted information. Polarity can be divided into two categories (positive and negative), 3 categories (positive, neutral, negative) or up to 5 categories (ranging from very negative to very positive). This type of analysis is very important to businesses to be able to gauge the popularity of products taking into account customer feedback in real time. This takes different importance when sorting data at large scales, because a business can, in real-time, be able to know how good their product is doing using consistent criteria. For example, these data can help with customer service [79] and deliver a better product to their clients, consequently being able to increase sales. This type of analysis is also used for image recognition [80].

2.4.2 Emotion Detection

The second type is Emotion Detection [81], which can detect sentiments like happiness, sadness, anger, surprise, among others. These emotions can give a more nuanced view on their products to a business, helping them to fix in a more precise way what they want to get better. It can also help businesses check the market sentiment on their products, and consequently help with their customer service. These emotions can also help to assess the cultural views on a topic [82].

2.4.3 Aspect Based Sentiment Analysis

The third type is Aspect Based Sentiment Analysis [83] which has as its objective to know what aspects are related to a sentiment. For example, in a product review where customers complain that the deliveries are slow, this type of SA must be able to tell them that the sentiment towards the reviews is negative for the aspect of delivery. However, it is important to mention that this can get more complicated when several aspects with different sentiments are in the same inputted text. Aspect Based Sentiment Analysis can give even more specific information about the popularity of their product, making it even easier to determine what the customers want to see changed and what is already working.

Aspect based sentiment analysis is also used for rating predictions [84].

2.4.4 Intent-Based Sentiment Analysis

The last type is Intent-Based Sentiment Analysis [85]. This type of SA is very helpful for businesses that want to know the intent of customers. This type of analysis can tell businesses if a sudden increase in product views may indicate an increase in sales or simply an increase in people browsing their products. This can help marketing departments to know what products to market better to increase sales.

2.4.5 Sentiment Analysis in Portuguese

SA tools and resources on the market nowadays, including Vader [86], TextBlob [87], SentiwordNet [88], among others tend to be very good for the English Language. Even though a lot of work has been put into this area, most languages are still falling behind. This problem is the one we are going to tackle in this dissertation.

Tools like Textblob and Vader have multilingual capacity but they tend to first translate from Portuguese (or any other language) to English and then use the English models to evaluate the inputted information. Even though this seems to be a good solution in principle, in practice this does not seem to obtain great results due to the state of translation tools.

Several papers and projects have tried to help further the Portuguese language case in the Sentiment Analysis area.

Projects like *Projeto Natura* from José João Almeida [89] is a good example of a project trying to further the fields of NLP and SA in the Portuguese language. It tries to further this cause by releasing helpful tools, dictionaries and corpora, among many other tools.

Another good project trying to further this cause is the Netlang project [90] [91], which focuses on finding online hate speech. Another area being explored for sentiment analysis is the social networks area [92].

The use of embeddings and transformers is the new big thing in SA. There are, already, a couple of projects that present us with pre-trained embeddings [93] and transformers [94] [95] [96] for the Portuguese Language. Projects like these are a fundamental part for the advancement of the Portuguese Language and the field of sentiment analysis, because it allows anyone to use them simply (often explained how to use with step-by-step guides by the authors).

2.5 Previous efforts and tools

There are many approaches to Sentiment Analysis but they can be assigned into 3 categories.

The first is a lexicon-based approach that uses lexicons and corpora of words that already have a sentiment assigned and we compare words in the inputted text with the words in the lexicon(or corpus) to see what are the most prevalent sentiments.

The second approach is an ML Based approach, where a labelled dataset is used with (at least) two fields, one for the inputted text and another for the sentiment associated with the text. Then, we create ML models with the aforementioned dataset as the training dataset; in the end, we get a model that can predict sentiment, where the quality of the predictions is dependent on the quality of the dataset, pre-processing and the model itself.

The last approach is a Hybrid Approach where we use both lexicons/corpus and Machine Learning.

In the next subsections, we will discuss the most used tools for each approach, starting with the most known lexicons and followed by the hybrid (or machine learning only) approaches on the market right now.

2.5.1 Lexicon Based Approach

The Table below (Table 1) shows some of the tools for the lexicon-based approach.

Name	*Authors*	*Year*	
LIWC	James Pennebaker et al	1993 - 2015	[97]
ANEW	Margaret Bradley, Peter Lang	1999	[98]
NRC	Saif Mohammad, Peter Turney	2013	[99]
AFINN	Finn Nielsen	2009	[100]
SentiWordNet	A. Esuli, F. Sebastiani	2006 - 2010	[88]
BING	Minqing Hu, Bing Liu	2004	[101]
Loughran	Tim Loughran, Bill McDonald	2011	[102]
SenticNet	Erik Cambria et al	2010 - 2020	[103]
SO-CAL	Maite Taboada, Jack Grieve	2004 - 2011	[104]
SemEval	Adam Kilgarriff	2002 - 2018	[105]

Table 1: Lexicons for Sentiment Analysis

As one can see, these lexicons, which started (in some cases) a long time ago, have been improved consistently. LIWC and SO-CAL are the most significant examples of the previously mentioned phenomena.

LIWC is updated every 6/7 years, while SO-CAL every 2 years has a new version out to keep being up to date.

Some of these lexicons are free to use (or free to use for educational/research matters), making it very friendly for researchers to use them. Being free to use means that even if the initial creator stops updating them, researchers can use their work to create better and more updated lexicons or even use the same methodologies to create similar lexicons for other languages.

The lexicons themselves work by having a set of words classified by sentiment. Most lexicons classify words by either positive or negative. Newer lexicons are starting to get a set of emotions, giving each word a set of emotions to get more specific results.

The sentiment outputted is normally calculated by the sum of the words present in the inputted text. It can also be calculated by non-linear functions giving weights to some emotions compared to others.

The biggest difference between these lexicons is the sentiments they are classifying. For example, LIWC has an output of 90 variables that give us information ranging from general descriptors to linguist dimensions to psychological constructs. This lexicon is probably the most complete but sometimes we do not need so many outputs. For example, if we want to know whether a phrase is positive or negative we can use a simpler lexicon like sentiwordnet. We may want to get more or less specific in our predictions and so we choose the lexicon to use accordingly.

Another thing we have to take into account is the way the lexicon entries were evaluated (in terms of the sentiments present in certain words). In some cases like LIWC, ANEW, NRC they were manually evaluated by several people using the same methodology. In other cases, it is manually evaluated by only one person like AFINN in which the author uses the same methodology as the ANEW lexicon to evaluate more words and create a new version of ANEW. The last case is evaluating using other tools and lexicons in an automatic way like some versions of Sentiwordnet. According to the methodology used it can better match our problem and give us better results.

In the case of lexicons translated to other languages we need to explore the possibility of certain words having different meanings (in terms of sentiments associated with them), this may incur unwanted mistakes.

2.5.2 Hybrid/ ML Approaches

In this section, we will tackle hybrid and ML approaches. We decided to join these two approaches into one section because, in most cases, the tools already created for machine learning are not open source code, and because of this, we may not know if they are hybrid or not. In most cases, these tools do have hybrid approaches.

The Table (Table 2) shows some tools and embeddings, and then we will discuss them.

Name	*Author*	*Year*	
Vader	C. Hutto, E. Gilbert.	2014	[86]
TextBlob	S. Loria et al.	2013	[87]
MonkeyLearn	Monkey Learn	2015	[106]
Rosette	Rosette	2015	[107]
Aylien	Aylien	2016	[108]
Glove	J. Pennington et al	2014	[109]
Word2vec	Tomas Mikolov	2013	[61]
C&W	Ronan Collobert et al	2011	[110]
SSWE	Duyu Tang et al	2014	[111]
Bert	Jacob Devlin et al	2018	[65]
ELmo	Matthew E. Peters et al	2018	[112]
Flair	Alan Akbik	2018	[113]
FLERT	Alan Akbik	2020	[114]

Table 2: Tools for Sentiment Analysis

In these approaches, we have three types of sub approaches embeddings and neural networks, transformers and applications.

The applications are Vader, textblob, monkeylearn, rosette and aylien. This is the easier way to evaluate, in most cases, we can have an application where we input the phrase to be classified and the program returns the answer. In this specific situation, anyone can classify a set of their phrases. The second way these applications work is by giving us access to an API where we can classify our set of data. There are several things to take into account when choosing the one to use. For some of these like rosette and aylien we will need a paid version to use, this may be a hurdle for some projects. If we were not a programmer, we may need to consider only these applications, because we may not know how to work without them otherwise. Another thing to take into account is that these applications (especially the paid ones) have complimentary extras that may be beneficial to our use case.

The next two approaches are very helpful to let us create our classifiers. With embeddings and NNs like SSWE, C&W, Glove and Word2vec we can have word embeddings trained specifically to our use case.

The last approach is more advanced than the two we have previously spoken of. Transformers are a cutting-edge field that emerged recently; and with transformers like BERT, FLAIR and ELMo we have a great way to start creating transformers. FLERT is another transformer, that emerged from FLAIR and BERT, that we can use. In general, the most important step is to choose wisely the tool to use according to the problem we want to solve and the resources we possess.

In ML approaches we usually create models that evaluate sentences for the predominant sentiments. Using a pre-trained model is typically faster than lexicon-based approaches to give outputs because the ”slower”bit of the process is already done before the users input their texts.

A good example of a hybrid approach is using lexicons to evaluate a set of sentences for their emotions and using the result as the training input for the model created.

These approaches, when done right, give us excellent results in a fast way.

Chapter 3

Development Approach

In this chapter, we will explore the problem we want to tackle and how we will tackle it.

We will delve into the architecture and guidelines created for Omnium AI, for the sake of creating an organised project that can be used by the company. Some details may be omitted for the protection of the company's Intellectual Property.

We will examine the technologies used in order to achieve a solution to our problem.

3.1 The Problem

This dissertation aims to give a path on how to approach a Sentiment Analysis (SA) problem in the Portuguese language.

The approach we are going to propose is subdivided into 3 major steps.

First of all, we will do an exploration of how to get data from different sources with the purpose of creating our datasets. We are going to evaluate the quality of the data by using different methods. We will delve into Exploratory Data Analysis (EDA), to summarise the main characteristics of these datasets by employing different visualisation methods.

Secondly, we will prepare different pipelines of Transformers for pre-processing to normalise the text our models will have as input. We are going to use different transformers and explain how we can use them for the Portuguese language with less commonly used tools.

Furthermore, we will show a simple way to use and create different models using these datasets. These models will range from simple Machine Learning (ML) models to more complex Deep Learning (DL) models.

Lastly, we will streamline this 3-step process by creating a package that will give its user an easy way to create SA models in Portuguese.

For an easy-to-use result, our Models and Transformers will be added to the company's API in accordance with their architecture and guidelines.

Throughout the next chapters, we will follow the proposed approach by explaining the aforementioned steps one by one, in detail.

Before exploring this approach, we will delve into the architecture we are going to follow and the technologies we used to reach our end goal.

3.2 Architecture

In this section, we will present an architecture that was used as a baseline to create sentiment analysis models.

We will start with an explanation of Omnium AI's architecture and the addition to their package.

3.2.1 Omnium AI - Architecture

Omnium AI created the following architecture (Figure 8), in order to achieve a tool that can create models for different applications. This architecture was inspired by the Autogluon architecture and some of the implemented methods are wrappers from Autogluon and other open-source libraries.

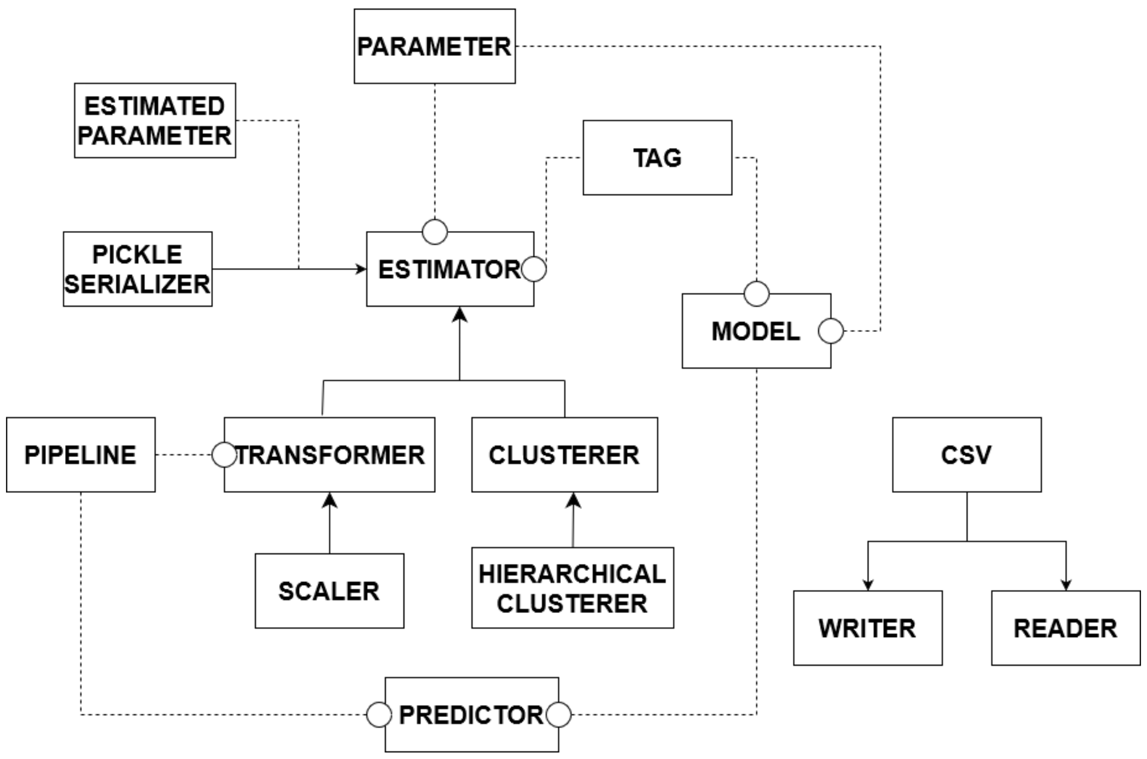


Figure 8: Omnium AI Architecture

This architecture allows the reading and writing of datasets with the CSV modules, also allowing the creation of Pipelines with Transformers and Predictors to process the data read and run it through an already trained predictor. More, it also allows the creation of several models with the data read and saves the predictors to use to evaluate data later.

This architecture is achieved in the following Python package called Omnia (Figure 9).

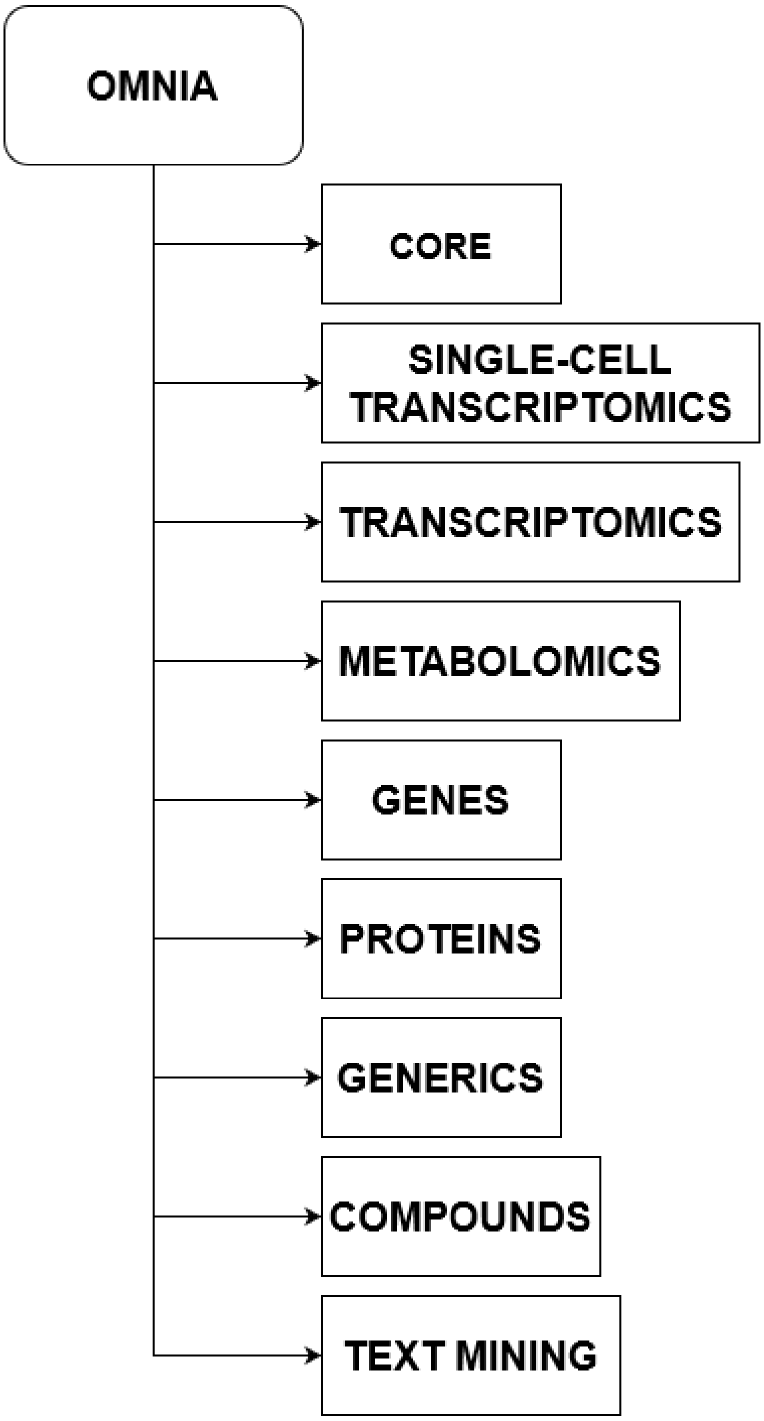


Figure 9: Omnia Package Composition

This package is composed of sub-packages for each project undertaken by Omnium AI.

For our project, we will focus on 3 sub-packages: Omnia-core, Omnia-generics and Omnia-Text-Mining. In the next figure (Figure 10), we can analyse these packages in a more in-depth way.

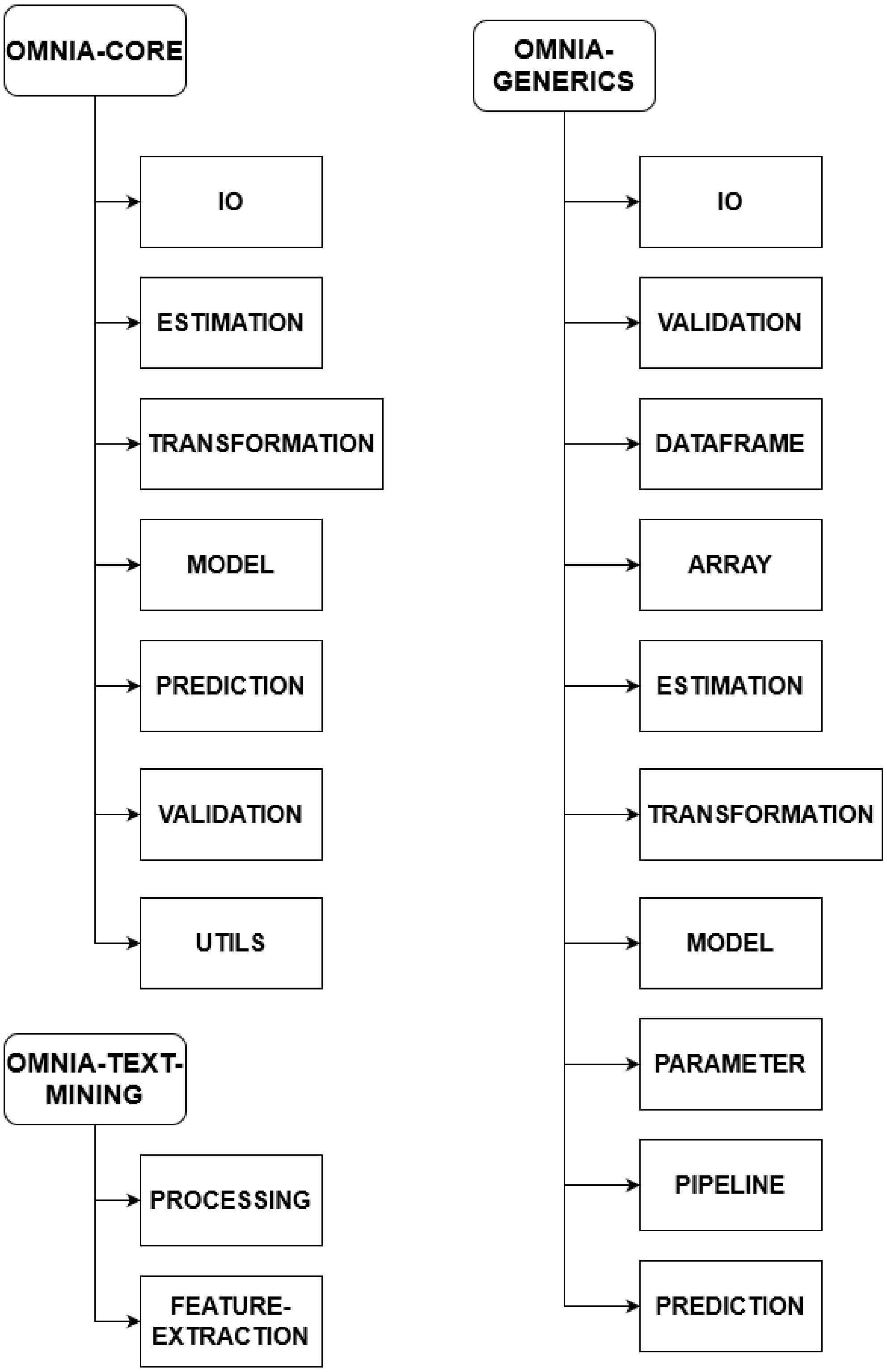


Figure 10: Omnia-Core, Omnia-Generics and Omnia-Text-Mining Package Composition

Omnia-Core is the core of the platform, that achieves the architecture above (Figure 8). It includes the several classes seen in the architecture. These classes are Estimation which regulates the Estimators, the IO handles the reading and writing of data, Model which controls the creation of models, Prediction which regulates the predictors, Transformation which handles transformers, Utils which controls the Serialisation and Validation used to regulate the TAG validation. Tag validation is a validation method that uses tags to validate data. Furthermore, Omnia-Core has the Pipeline class that handles the creation of pipelines of transformers and predictors.

These classes are abstract classes that abide by an object-oriented way of programming.

Omnia-Generics is the sub-package that gives a compilation of useful classes that may be common to different sub-packages in Omnia.

Omnia-Generics gives us a way to read, write and store data in Array, Dataframe and IO. Even though we are only using CSV and pandas dataframes we can read and write data in JSON and YAML. We can also work with other types of dataframes that are not Pandas.

Omnia-Generics gives us a couple of pre-processing methods created as Transformers in Encoding, Scaling, and Transformers.

Omnia-Generics has a couple of models and predictors in Models and Predictors, that are generally used by several other projects.

Omnia-Generics has a setup and pipeline class so that the different projects can create their pipelines in a standardised and streamlined way.

Omnia-Generics also gives us several standard tags created for the tag validation.

It is in this package that we are going to code our models and the pre-processing methods that can be used by other projects, while having access to materials created by other Omnium AI projects.

The last package, Omnia-Text-Mining, was created for our specific project, which includes 2 dissertations: ours and Tiago Silva's dissertation. It is here that we will create all the needed pre-processing Transformers, that are specific to our Text Mining (TM) projects.

All the code provided to these packages follows a strict set of rules created by Omnium Ai. These rules will be explored in the next subsection.

3.2.2 Omnium AI - Guidelines

In this section, we will delve into the set of rules created by Omnium AI.

These rules help to maintain a standard so that everybody can easily understand the projects and make future changes.

The most important rules are the creation of a requirements file to avoid import version problems; it is asked that every package follows the same package Hierarchy that includes a README file explaining the package objectives, the requirements for the aforementioned package (and the versions needed), a setup configuration file, the source code and their unit tests and an example using the package. This Hierarchy makes the comprehension of the package easier.

There is a strict guideline for imports to avoid circular import errors.

The code itself needs to have all its functions typed and documented using Python Docstring. This documentation must have an explanation of the inputs, outputs, the objective of the code and an example of the code being used.

To submit our code to a package or to create a package we use Version Control System where for each task we code we create a branch. When a task is finished, a pull request is created for an Omnium AI member to evaluate the solution to the task. If the solution is satisfactory, according to the company's standards, the branch is merged into the development branch until another version of the project is released. When a new version is released, the developed branch is merged into the main.

3.3 Technologies

In this section, we will go into more depth on what technologies we are going to use.

Our project was developed using Python and some of its open-source packages.

The main packages used in this dissertation were:

1. **Numpy** [115]: Package fundamental to computer science, provides an assortment of routines for fast operations over arrays;
2. **Emoji** [116]: Package that converts Unicode emojis into emoji names in a given language;
3. **Spacy** [117]: Package that offers many methods for Natural Language Processing;
4. **NLTK** [118]: Package that deals with text, providing methods to perform, among others, tokenization, lemmatization and part of speech tagging;
5. **Autogluon** [119]: Package that enables an easy-to-use Automatic Machine Learning tool with a focus on creating models;
6. **Pytorch** [120]: Framework that allows an easy implementation of several Deep Learning models;
7. **Tensorflow** [121] and **Keras** [122]: Framework that allows an easy implementation of several Deep Learning models;
8. **Scikit-learn** [123]: Package that offers simple and efficient tools for predictive data analysis;
9. **Pandas** [124]: Flexible package that offers a straightforward data analysis and data manipulation tool;
10. **re** [125]: Python library that helps to deal with regular expressions;
11. **Google Translate** [126] and **OpenNMT** [127]: Packages that provide a simple way to translate text.

12. **Twint** [128]: Advanced scrapping tool that allows, among others, the scrapping of Tweets;
13. **MatPlotlib** [129]: Library that helps to visualise data.

We will also use available resources in Omnium AI's Omnia package.

The usage of these packages helps us achieve a more efficient product. Without open-source projects, like those aforementioned, it would be a very challenging job to create tools like the one being created in our dissertation.

Summary

In a nutshell, the objective of this dissertation is to give us a stepping stone into SA and give Omnium AI a tool to create ML and DL models.

We will be able to create pipelines that generate models, for a specific project we are tackling simply and concisely.

These Pipelines are a combination of a Reading step where we will read the data, a pre-processing step where we will use Transformers to normalise text and the last step where we generate models according to our data. These models can, subsequently, be used to evaluate text on the same topic as the data that created the model.

By creating more than one model, we will be able to compare results on our projects choosing the best models and the best pre-processing for our data so that we have the best possible models for our projects.

We will also have an idea of the next steps we may take to get other models that may be good for our projects.

To achieve our goals, we can use an architecture like Omnium's, using the same rules and packages they did, to create a tool that can be used effortlessly.

In the next chapter, we will give a detailed explanation of how we tackled the aforementioned approach, offering a thorough description of all 3 steps we articulated previously. We will start by exploring datasets, followed by Transformers and finalising with Models.

Chapter 4
Development

In this chapter, we will discuss how to develop a 3-step architecture. We will start by exploring how to gather data, followed by a breakdown of different methods to transform data and ending with an exploration of the models we created.

4.1 Datasets

This section will start by explaining how we can gather data, which stands as the first step to be able to create SA models.

There are two methods for collecting data. The first method to collect data is to start with pre-built datasets. There are various sources that provide us with free-to-use data, such as Kaggle [130], Data Driven[131], Crowd Analytix [132], and CrowdAI [133].We used the Kaggle platform to collect data. The second method is to create the dataset from scratch. It is common to collect social media posts, we utilised Twint to collect Twitter posts and create a dataset [134]. Either way, there is a need to ascertain if the dataset belongs to our project topic. The best way to substantiate this is by doing an EDA.

Sentiment must be defined before searching or creating datasets for SA. For this project, sentiment is defined as positive or negative connotations of text. With this, we can proceed with the search for a dataset. We found a couple of results for our topic of Covid19, with the best version being a dataset in Portuguese with the same definition of Sentiment as us, which we will explore in the next chapter.

Several datasets within our topic were in another language, we could have used translation resources to transform these datasets into viable data for our project. The results of the translations may not be good enough ultimately, but it's a path that could have been taken. Sources like OpenNMT [127] is a good approach to translating our data, through the use of pre-trained models. As we are using python, the Translators' package [135] is another useful alternative that lets us choose several sources to use to translate our text.

Once we have exhausted these paths, we can always create our dataset from scratch using Twint. Twint allows us to retrieve posts from Twitter according to a set of filters. This approach would imply manual labelling of the posts to get the best possible dataset. This step is very important, without a well-labelled dataset we can not get a good model. We created a small dataset from this source as a sample of what can be done, we will explore this dataset in more depth in the next chapter.

As previously mentioned, we got a dataset from Kaggle and created a sample dataset using Twint. For every dataset, we checked if it is adequate for our project by doing an EDA. Our EDA starts with an analysis composed of feature extraction methods where we find out the words that occur more in our data. Methods like Bags of words, Wordcloud, Tf-IDF, word counts and common terms are all good options for this analysis. These methods are important because they provide an idea of what is being referred to in the data. For these methods to work best we removed stopwords.

This analysis was followed by an intersection of words from different labels in order to ascertain if there are words that are more common to a certain label. This analysis can give us extra insights into the data and the quality of its labelling.

To get a good model, the data should be unbiased. For this, we should balance the dataset, having more or less the same number of entries for each label. We do not need to worry about the size of the dataset because a bigger dataset does not mean a better model.

In order to normalise the text, we took out duplicate values because they do not give any extra information. Missing or null values in the labels were taken care of, either we manually checked each value and classified it with the correct label or we removed the entry from the dataset.

If the results of the model we created are not good, the first point we will come back to is the quality of our labelling and the width of information in our entries. In these cases, we may need to manually check the quality of our labelling and our entries. If our entries are not good, we may need to scrap the dataset and start fresh and a new set of data.

4.2 Transformers

After gathering the data, we need to normalise it. This chapter describes various methods for the pre-processing of the data. The normalisation will reduce the amount of information the model has to learn, hopefully improving its efficiency. Even though the normalisation of text is very important, there is no correct set of methods we can use every time for optimal results. For each situation, we will test what methods help us achieve a better model and what methods make us lose knowledge and give us worse models.

Throughout the next section, we will explore a couple of techniques to normalise text. These techniques are staples in pre-processing for Natural Language Processing (NLP) and Artificial Intelligence (AI). In the last section, we will explore the approach to pre-processing we will employ to evaluate different pipelines.

4.2.1 Transformers Created

There are two main ways to normalise text. We can either replace parts of the text with more ”standard”alternatives or we can remove it altogether. For each situation, we have to evaluate what is the best transformer to achieve the best result. Across the next subsections, we will explore some normalisation problems and our solutions to them.

Convert Case

The first problem we are tackling is Letter Case. Since models are case insensitive, the same words written differently are seen as different entries. So it is normal to normalise the text by either converting it to uppercase or lowercase.

We created a lowercase transformer to solve this problem. By using the lowercase method from the String package [136], we easily convert our text into a lowercase version of it. There is also a method to convert to uppercase. This Transformer helps normalise the text. Still, in some languages, misplaced uppercase letters are used to express irony, anger or other sentiments. In these cases, using this Transformer may lose important knowledge our model could learn.

Tokenization and Lemmatization

The second problem we are addressing is the variation of some words. In languages like Portuguese, some words have a lot of variations. For example, nouns vary from a gender and a plurality point of view; verbs vary in terms of the tense being used. In some cases, these variations do not add any knowledge to our model.

There are several solutions to this problem. In our work, we solved it by using tokenization followed by a lemmatization of each token gathered.

This solution helps reduce the amount of variability of information the model has to learn. We achieved this solution by using Spacy [117] and NLTK [118]. Spacy provides us with the ability to lemmatize and tokenize text by using the NLP function.

Another solution commonly employed is the utilisation of tokenization followed by stemming. The main difference between stemming and lemmatization is that lemmatization replaces a word with the lemma of that word. Stemming replaces the word with a word with the variable part cut. For example, the word ”studies”is replaced with ”study”in lemmatization, and it is replaced with ”studi”in stemming. We can achieve this solution by using NLTK [118]. NLTK presents a stemmer function that stems words in a given language, including Portuguese.

Special characters

The third problem we are solving is how to deal with special characters. Characters like emojis and tags are common in datasets created from social media scraping. There are two paths to dealing with these characters, we can either remove them or replace them with their meaning.

For emojis we can replace them with a standard that encapsulates what the emoji is trying to convey. The Emoji package [116] is a Python package that converts Unicode emojis into a string with the name of the said emoji. This is a good solution to this emoji problem.

For name tags, we can just get the name and remove the tag character. If the tag is not giving us any knowledge we can remove the whole tag altogether.

For hashtags, we can remove the tag character and try to separate the different words. This process can be very computationally expensive and sometimes it does not compensate for the effort. This is our case, so we can either remove it or count it as a single word. For these two problems involving tags, we used regular expressions to remove the tags.

There is another solution for this problem that involves the segmentation of the tags to get the information present in the tags. Packages like Ekphrasis [137] do this for the English language and can be a stepping stone to creating a similar project to segment tags in Portuguese. Research papers like [138] that can help to create a tool for hashtag decomposition are also useful for tag segmentation.

Our solution to the tag problem was to remove the tags altogether.

Stopwords

The fourth problem we are confronting is stopwords. Stopwords are words that are considered irrelevant to the text. These words are frequently used in texts but do not offer any meaning. Prepositions, articles and conjunctions are good examples of stopwords. In most cases, stopwords do not add value and can be eliminated from the text without losing any information. NLTK [118] package offers a set of stopwords for 20+ different languages, including Portuguese. We use this set to remove these stopwords from our texts.

Unnecessary Details

The fifth problem we are facing is what to do with unnecessary details. Details like double spaces, new lines, links, URLs and punctuation are sometimes unnecessary information to the text and can be removed. This problem can be easily solved by using regular expressions. These regular expressions catch the information we want to remove and replace it with either a space, a standard or with nothing at all.

In problems like this we should try to evaluate if any of this information, which normally is unnecessary, truly can be removed without losing knowledge. Punctuation, for example, can give important information in some cases. In our work, we removed these unnecessary details.

Unnecessary Entries

The sixth problem we are handling is how to treat unnecessary or wrong entries in our data. We normally remove unnecessary entries like columns that are not being used. Another type of unnecessary entry are duplicate values, which do not add any informational value and therefore can be eliminated from the dataset. Entries that have a null label, NaN or an invalid label are examples of other unnecessary entries that can be removed or, if possible, changed to the correct label. This change should be manual to guarantee the quality and integrity of the dataset.

As our data is being stored in a pandas dataframe, the aforementioned changes can be efficiently done by using pandas methods. We removed duplicate values and manually replaced invalid or null labels in this project.

Typecasting

The seventh and last problem we treated was typecasting. Every bit of data that is a yes or no, true or false or something we can transform into a numeric value should be typecasted as an integer (or other numeric types). Numeric values are easier to understand for our models and so if we can change nominal data to numeric we should do it, especially if it is the label as it is easier to cast into a numeric value.

In our case we translated a positive/negative label into a 1/-1 label.

Other Problems

There are many more normalisation methods we could have explored. Methods like accent removal, the substitution of contractions, the transformation of numerals into their number counterparts, the replacement of numeric values into their type, the normalisation of acronyms and abbreviations, normalisation of date formats, phone numbers and other important numbers among many other transformations that can be specific to different situations.

These problems can be specific to some languages and to some specific problems. For each project, we may have specific normalisation steps we can do and that's why it is very important to study the area of our problem before starting.

4.2.2 Transformers package and Pipelines

After exploring the aforementioned transformers, we created the Transformers we needed for our specific use case and that Omnium AI may need in the future. There are some Transformers already present in the Omnia package that we are not going to use and so we are not going to discuss them.

The following figure, (Figure 11) shows us the Transformers we created as solutions to the aforementioned problems. These Transformers are going to be used to create our transformers pipelines.

In order to streamline the pre-processing we created Pipelines that let us use the various Transformers. These pipelines will normalise our texts in different forms and will output a new dataset with the text normalised. With these resulting datasets, we will create different models that we will use to evaluate which pipeline is best for our specific use case. The objective is to start with a pipeline with no transformers to get a baseline. We will then add different transformers going from the most general transformers like

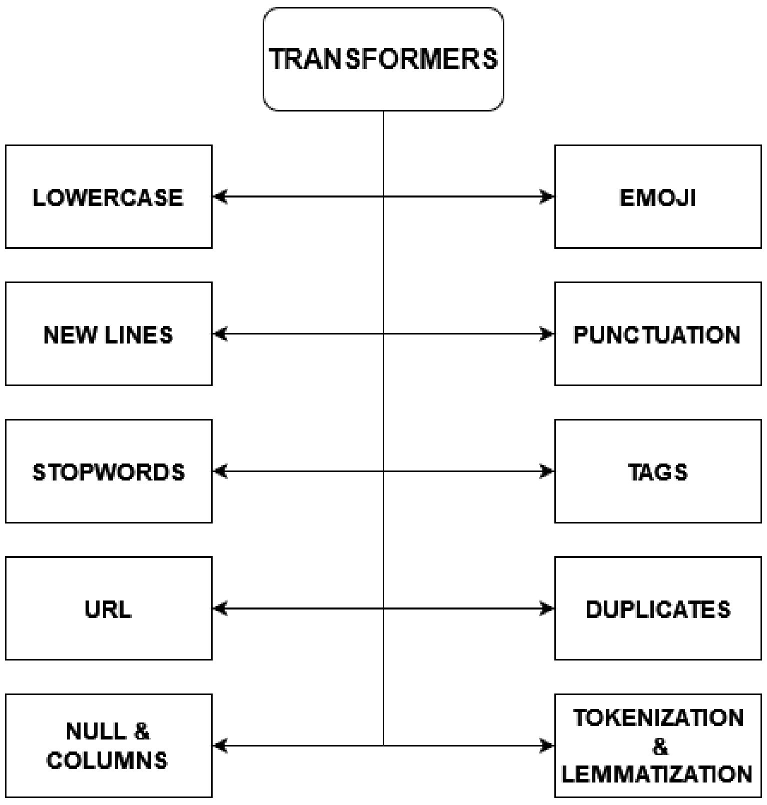


Figure 11: Transformers created for the package Omnia-Text-Mining, according to the aforementioned Transformers

removing duplicates, URLs, etc; to the more specific Transformers like removing stopwords, tokenization & lemmatization among others.

In the next chapter, we will go more in-depth into the pipelines created.

4.3 Models

After having gathered data and normalising it, we will focus on the models. In this section, we will delve into the models we created starting with ML models followed by DL models and finishing with Autogluon.

The ML and DL models present in this section were explained in more detail in the State of the Art chapter.

4.3.1 Machine Learning

In this section, we will highlight the ML models we will use.

In the following table (Table 3), we will display the machine learning models that we are going to use and their scikit-learn function [123].

Model	Function
Naive Bayes	GaussianNB()
Multinomial Naive Bayes	MultinomialNB()
Random Forest	RandomForestClassifier()
Bagging Naive Bayes	BaggingClassifier(model= GaussianNB())
Bagging Multinomial Naive Bayes	BaggingClassifier(model= MultinomialNB())

Table 3: Machine learning models and their Scikit-Learn functions

4.3.2 Deep Learning

In this section, we will delve into the different DL models we will use.

Recurrent Neural Network

Recurrent Neural Networks (RNN) is a type of Neural Networks (NN) that can be used for sentiment analysis. Our specific RNN has 3 unique layers, an embedding layer, an rnn layer and a dense layer. The first layer is an embedding layer with no pre-trained embeddings. The second layer is a SimpleRNN layer with a hyperbolic tangent function (tanh) as its activation function. The last layer is a fully connected dense layer with a sigmoid function as its activation function.

The model will use binary cross-entropy as its loss function, adam as its optimizer, and it will use F1 score as its primary metric.

Convolutional Neural Network

Convolutional Neural Networks (CNN) are NN that use convolutional layers. Our specific model has 3 unique layers, an embedding layer, a convolutional layer and a linear layer. The first layer is an embedding layer that uses a word2vec embedding. The word2Vec embedding was trained from the dataset being applied to the model. The second layer is a list of convolutional layers, that use the hyperbolic tangent function as its activation function. The last layer is a linear layer that uses the softmax function as its activation function.

The model will use binary cross-entropy as its loss function, adam as its optimizer, and it will use F1 score as its main metric.

Long Short Term Memory

Long Short Term Memory (LSTM) is a specific type RNN. Our specific LSTM has 5 layers, an embedding layer, an lstm layer, a dropout layer, a linear layer and a sigmoid layer. The first layer is the embedding layer that has no pre-trained embeddings. The second layer is the lstm layer which uses the hyperbolic tangent as its activation function. The third layer is a dropout layer with a rate of 0.3. The fourth layer is a fully connected linear layer. The last layer is a sigmoid layer that applies the sigmoid function to the output of the fully connected linear layer.

The model will use binary cross-entropy as its loss function, adam as its optimizer, and it will use F1 score as its main metric.

4.3.3 AUTOGLUON

Autogluon [119] is a package that gives an easy-to-use AutoML with a focus on ML and DL models. Autogluon is separated into several sub-modules specialising in text and tabular prediction as well as image and object detection. We will use the Tabular Prediction and Text Prediciton sub-modules to create ML and DL models.

The Tabular Prediction sub-module can create the following models: LightGBM, LightGBM Large, LightGBMXT, Catboost, XGboost, WeightedEnsemble and NeuralNetTorch.

LightGBM, LightGBMXT and LightGBM are ML models created by Microsoft. They are based on decision trees and have their focus on performance and scalability. Catboost is a ML algorithm created by Yandex researchers. They are based on gradient boosting for decision trees and have their focus on categorical data. This model prides itself on having good results without parameter tuning, providing good accuracy in a fast and scalable manner. XGBoost is also based on tree boosting. It provides a parallel tree boosting that solves problems in an accurate and fast way. NeuralNetTorch is a DL model that uses the PyTorch EmbedNet as its main layer.

The Text Prediciton sub-module can create a DL model named HFAutoModel and has the capability of applying pre-trained models and embeddings to this model. HFAutoModel is a Deep Learning model that uses a pre-trained model or embedding as a basis followed by a Linear layer. We will use this model and a version of this model that uses a pre-trained Bert embedding created by NeuralMindAi. This NeuralMindAi Bert embedding is a pre-trained embedding specifically trained for the Portuguese Language.

Autogluon lets us use any embedding we have created or that is present in Hugginface. Hugginface [139] is an AI community platform that specialises in providing tools to use for AI. This embedding and many others, for a diverse amount of languages, are provided on this platform.

The models in these sub-modules will be used in comparison to the models described in the previous sections.

Chapter 5

Results & Discussion

In this section, we will explore the use case we chose to compare results. We will delve into the datasets we used and created for our use case. We will dig into the transformers and the combinations we will utilise to normalise our datasets. We will finish with the results our models achieved and a small discussion about the results.

5.1 Use Case approach

5.1.1 Use case

With the purpose of evaluating the methods we explored in this section, we will create a use case. The use case chosen is the Portuguese-speaking population's sentiment towards Covid19. We will collect and create datasets for our use case. We will process the data using the transformers we created. We will build models for the specific purpose of evaluating the sentiment toward our use case.

For our use case, sentiment will be considered as positive and negative, this can be easily changed for other interpretations of sentiment. By using the methods explored in the Datasets section in the Development chapter, we will use Kaggle [130] to obtain a dataset for our purpose. We used Coronavirus Twitters NLP: Text Classification (PT) [140] dataset for our purpose. We will also utilise a dataset created in Twint [128] as a sample dataset for the creation of datasets starting from scratch, we will explore in more depth this dataset in the Datasets section.

5.1.2 Approach

Our approach to the abovementioned Use Case can be divided into 4 steps.

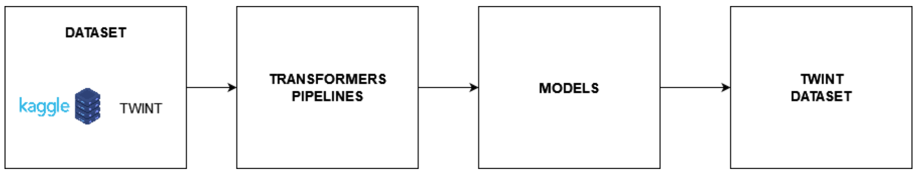


Figure 12: Use Case Approach

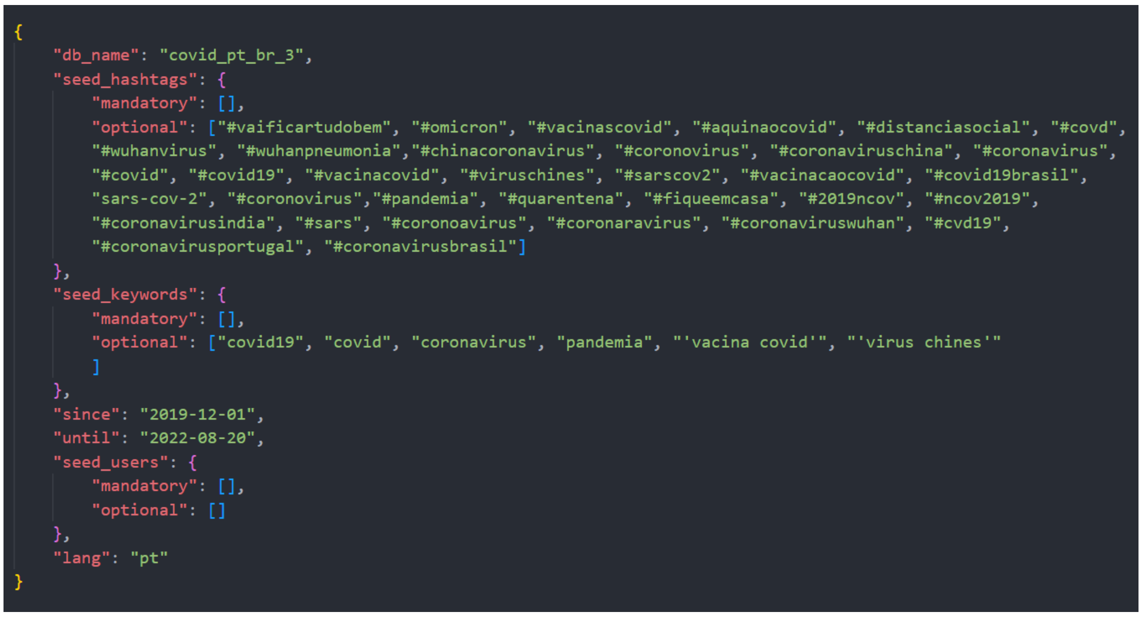
As we can see in the figure above (Figure 12), we collect two datasets and create a set of pipelines to normalise the text in these datasets. After normalising the text, we apply the Kaggle dataset to our models that are composed of ML, DL and Autogluon models discussed in the last chapter. In the end, we will take the best combinations of transformers and models and apply them to the Twint sample dataset. In the subsequent sections, we will discuss each of these steps.

5.2 Dataset

Following the abovementioned approach, this section will explore the datasets. We will do an EDA to ascertain the quality of the dataset.

We will start by explaining how we obtained the Twint sample dataset. This sample dataset was created by Constança Elias for her dissertation named Twitter Observatory: developing tools to recover and classify information for the social network Twitter [134]. This dataset is created with her API that uses Twint to extract data from Twitter. The extraction receives a set of options for the process to start and returns a dataset labelled with 1 if the entry is correctly classified as being part of the topic wanted otherwise returns 0. The user gives a set of keywords and hashtags for the topic of the dataset. If the tweet scrapped has either one of the keywords or hashtags provided by the user, the entry will be labelled as a good entry (being labelled as a 1), if not it will be classified as an entry that has nothing to do with the topic requested (being labelled as a 0). The user can also input the language for the tweets, and a date interval for the tweet. After this process we have a resulting dataset labelled for its relevance against a pre-defined topic.

The figure below (Figure 13), shows us an example call for the aforementioned API.



```
{
    "db_name": "covid_pt_br_3",
    "seed_hashtags": {
        "mandatory": [],
        "optional": ["#vaificartudobem", "#omicron", "#vacinascovid", "#aquinaocovid", "#distanciasocial", "#covd",
        "#wuhanvirus", "#wuhanpneumonia","#chinacoronavirus", "#coronovirus", "#coronaviruschina", "#coronavirus",
        "#covid", "#covid19", "#vacinacovid", "#viruschines", "#sarscov2", "#vacinacaocovid", "#covid19brasil",
        "sars-cov-2", "#coronovirus","#pandemia", "#quarentena", "#fiqueemcasa", "#2019ncov", "#ncov2019",
        "#coronavirusindia", "#sars", "#coronoavirus", "#coronaravirus", "#coronaviruswuhan", "#cvd19",
        "#coronavirusportugal", "#coronavirusbrasil"]
    },
    "seed_keywords": {
        "mandatory": [],
        "optional": ["covid19", "covid", "coronavirus", "pandemia", "'vacina covid'", "'virus chines'"
        ]
    },
    "since": "2019-12-01",
    "until": "2022-08-20",
    "seed_users": {
        "mandatory": [],
        "optional": []
    },
    "lang": "pt"
}
```

Figure 13: API call

This call to the API was made several times with different date intervals and the outputs were joined into a single dataset (with more than 2M entries).

After getting this dataset we created a new dataset by using a random sampling technique. We sampled 100 entries (relevant to the topic) which we then manually labelled for SA. Finally, we got a small dataset manually labelled for SA. Throughout the next subsections, we will do a small EDA on this dataset.

The other dataset we are going to use is the dataset obtained through Kaggle, we will call it the Kaggle dataset. The Kaggle dataset was created by Francielle Vargas [141]. This dataset has 600 entries classified in accordance with their polarity (positive, negative classification). For the classification, the author tested a cross-domain strategy to measure the performance of the classifiers among different domains.

Throughout the next section, we will do an EDA on both datasets.

5.2.1 Exploratory data analysis

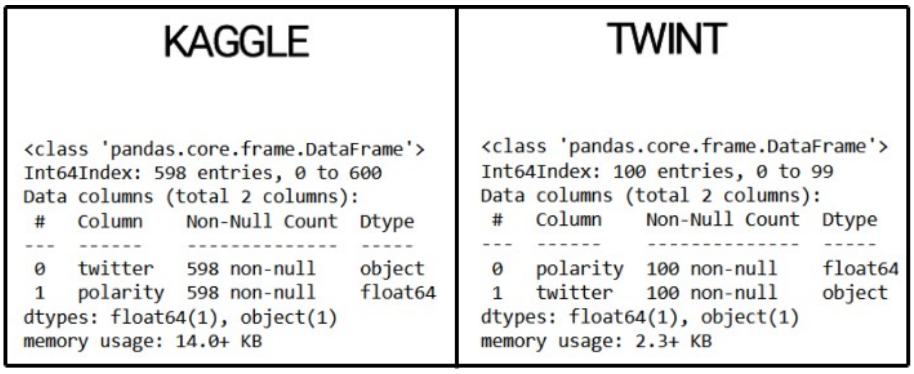
We will start by delving into what preparation we did for the datasets, followed by an exploration of the datasets in terms of balance, size and common terms.

Data Preparation

Before we begin the data exploration, we initiated the removal of unnecessary data. This unnecessary data were columns we were not going to use, duplicate entries, entries with null values as labels, stop-words, name tags and hashtags. We also tokenized, lemmatized the text and replaced Unicode emojis with a standard that encapsulates their meaning.

General information on the dataset

After having prepared the data, we proceeded to gather the general information of the dataset. The following figure (Figure 14) gives us information on the columns present in our datasets, their types and the number of null values.



KAGGLE

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 598 entries, 0 to 600
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   twitter   598 non-null    object
 1   polarity  598 non-null    float64
dtypes: float64(1), object(1)
memory usage: 14.0+ KB
```

TWINT

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   polarity  100 non-null    float64
 1   twitter   100 non-null    object
dtypes: float64(1), object(1)
memory usage: 2.3+ KB
```

Figure 14: Description of the dataset

In our datasets, we have 2 columns, a float 64 column named polarity without null values and an object column named twitter without null values. The polarity column will be our label which represents the Sentiment present in the text and the twitter column will be our text.

The following tables (Table 4, Table 5), gives important information on columns with numeric values.

	Count	Mean	Std	Min	25%	50%	75%	Max
Polarity	100	0.22	0.980414	-1	-1	1	1	1

Table 4: Description of the polarity column in the Twint Dataset

	Count	Mean	Std	Min	25%	50%	75%	Max
Polarity	598	-0.003344	1.000832	-1	-1	-1	1	1

Table 5: Description of the polarity column in the Kaggle Dataset

In our datasets, we only have 1 numeric column and the tables above give us the count of entries present in the dataset, the mean of values in all entries, the standard deviation, the min and max value and the first, second e third quartiles.

Balance of the dataset

Now that we know the general information about our datasets, we will examine the balance of the datasets. This step is very important because an unbalanced dataset introduces a bias towards the label that has more entries. The following Figure 15, displays the balance of entries for each of our labels, in both datasets.

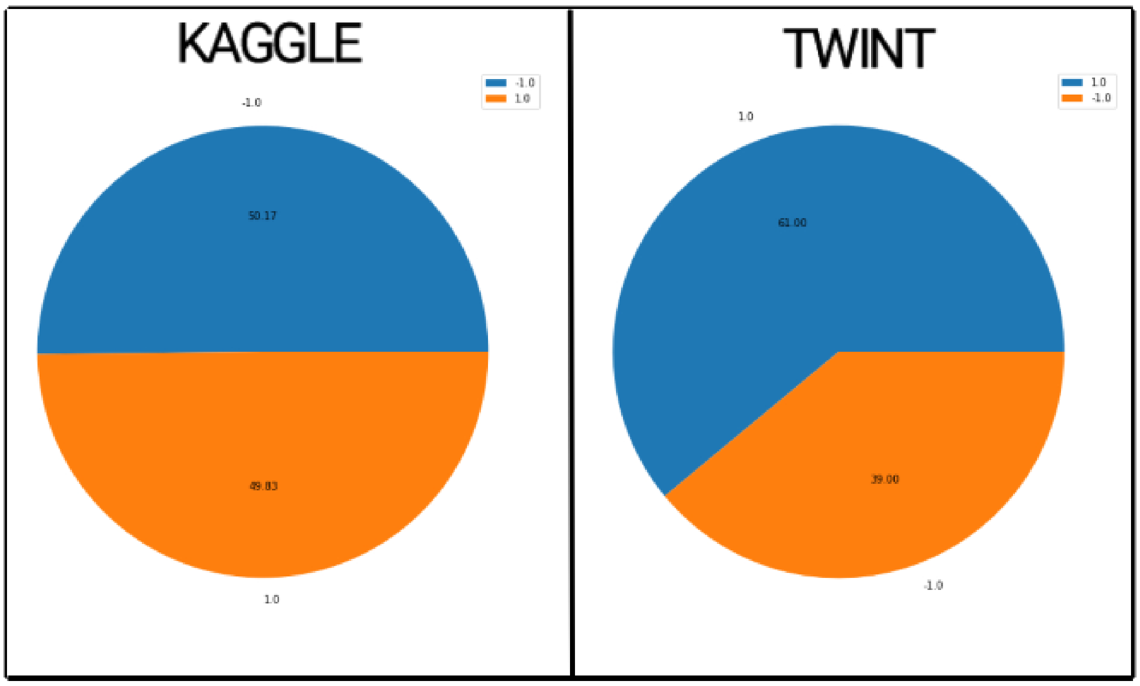


Figure 15: Balance of the dataset

As we can see, the Twint dataset is slightly unbalanced towards the positive label, which may be a problem, specifically for smaller datasets like this one. On the other hand, the Kaggle dataset is almost perfectly balanced.

Common Terms of the dataset

After collecting general information about the datasets and checking their balance, we need to check the quality of the dataset in terms of labelling. In this first analysis, we will explore the most common terms for each label.

The Figure below (Figure 16), shows us the common terms for the positive and negative labels from each dataset.

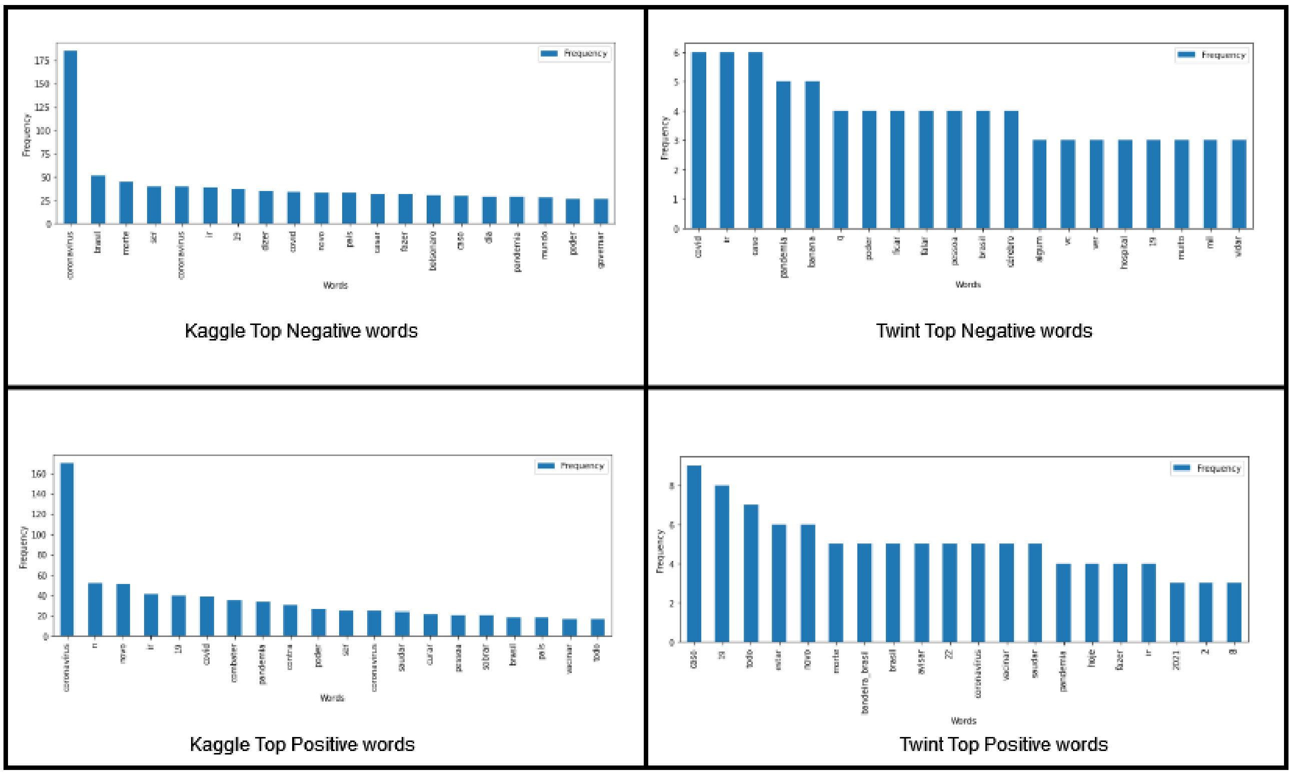


Figure 16: Top words from our datasets

The figure (Figure 16) suggests that the entries from the dataset seem to be good and related to our Covid19 topic. Words like *"pandemia"*, *"covid"*, and *"coronavirus"* (which translates to pandemic, covid and coronavirus) are prevalent in every label which indicates that both datasets are related to our Covid topic. Even though these images provide a good overall picture of the dataset, we will now intersect the words from positive and negative entries to get the top words specific to each label.

The figure below (Figure 17), displays the aforementioned intersection for the Kaggle dataset. The upper graph represents the words prevalent in the positive label and the lower graph represents the words from the negative label. Words like *"curar"*, *"vacinar"* and *"combater"* (which translates to cure, vaccinate and combat) are good examples of words that seem to indicate a positive connotation, while *"morte"* (which translates to death) is a good example of a word that indicates a negative connotation.

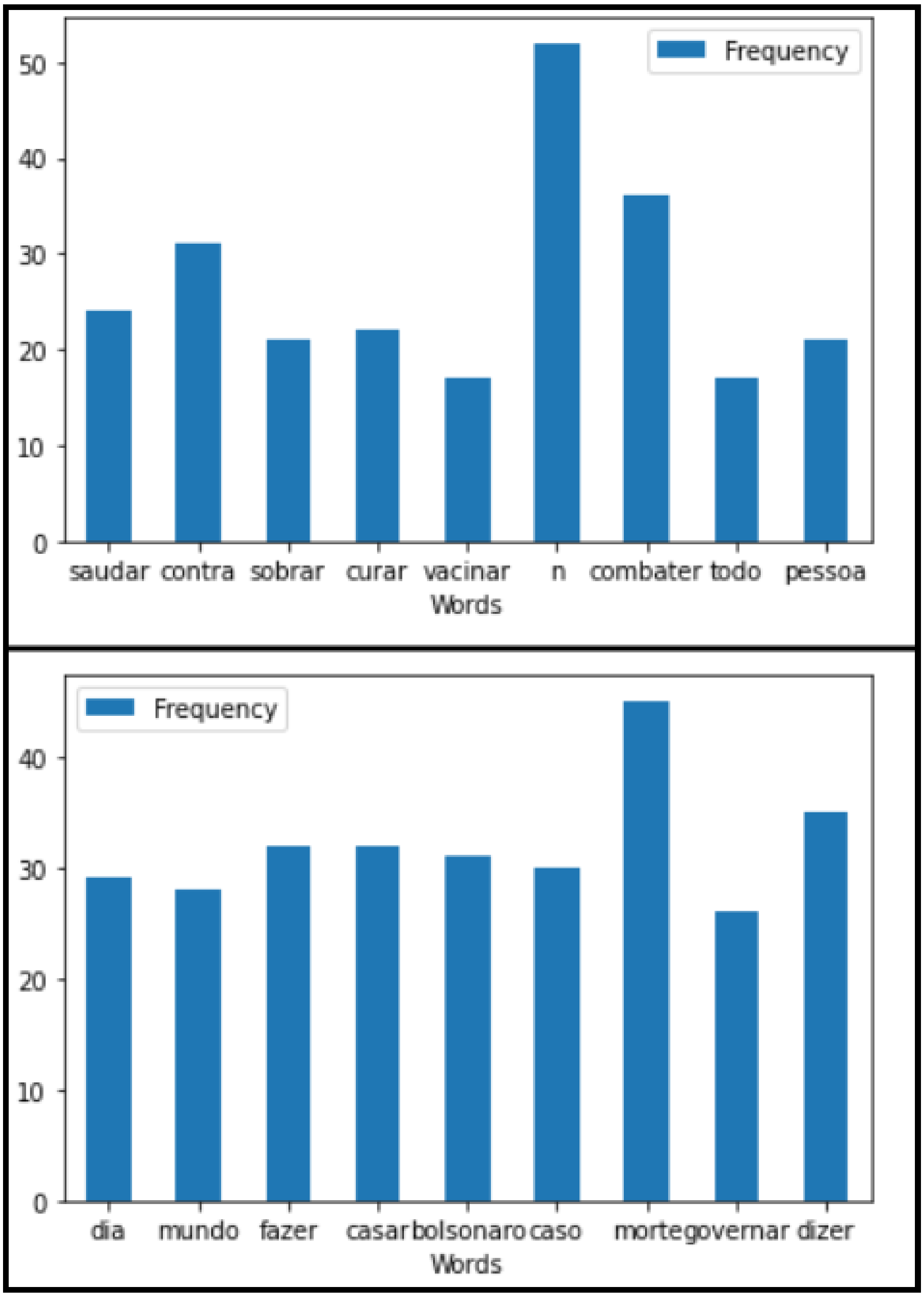


Figure 17: Top words from each specific label in the Kaggle dataset

The next figure (Figure 18) represents the equivalent graphs for the Twint dataset.

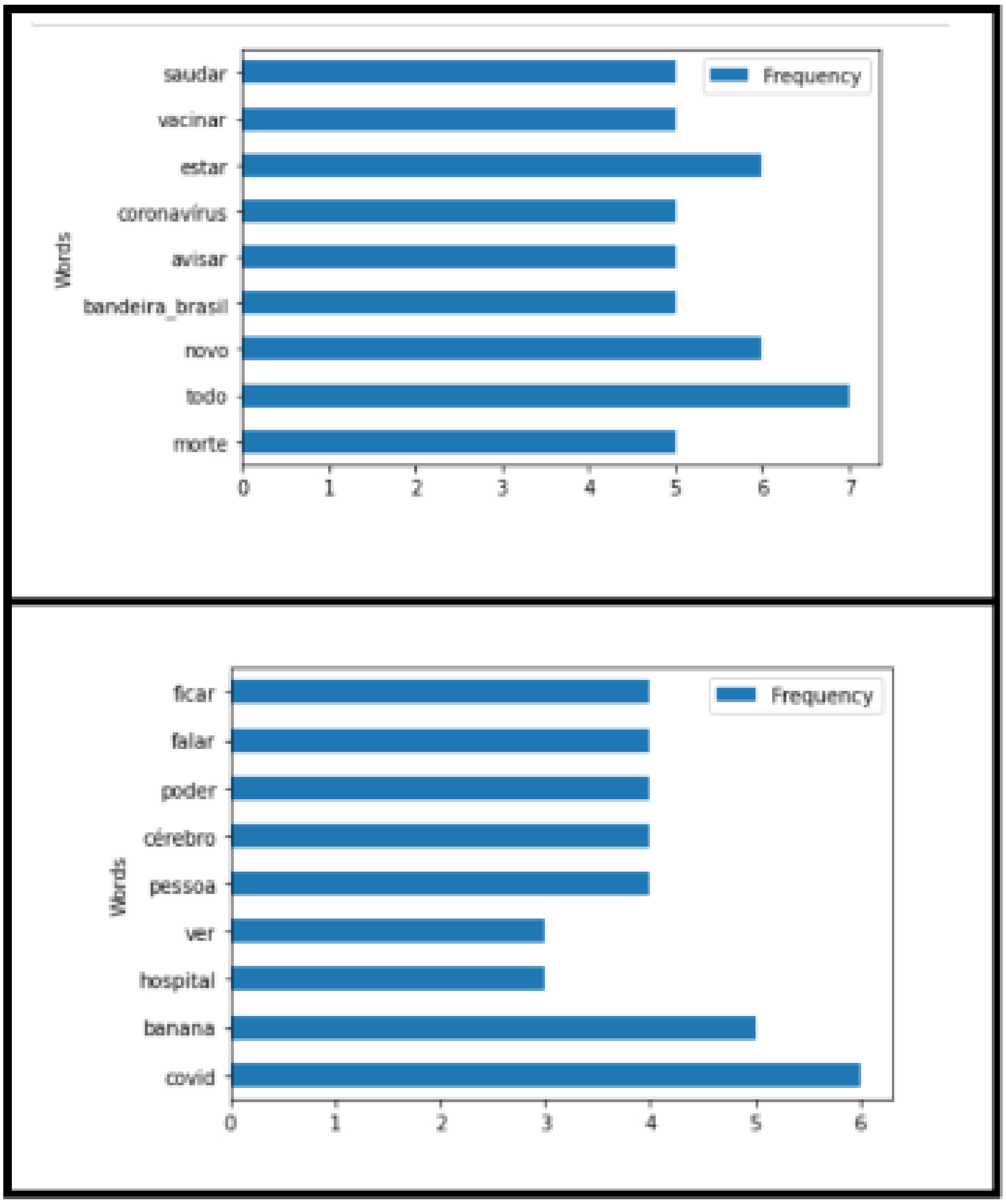
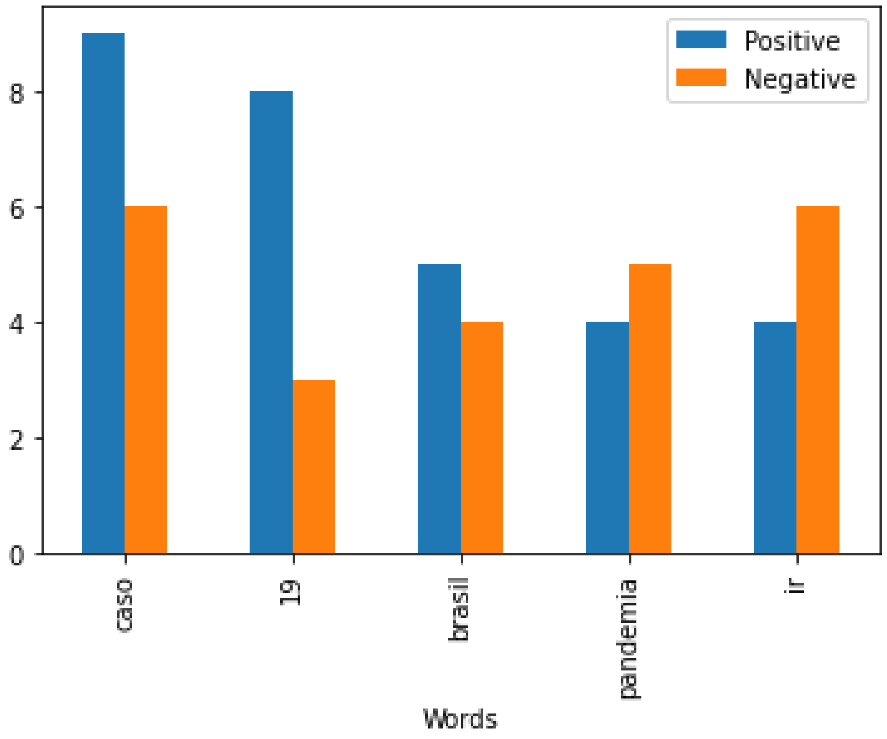


Figure 18: Top words from each specific label in the Twint sample dataset

As we can see words like *"vacinar"* (which translates to vaccinate) is normally related to a positive sentiment of fighting the Covid19 virus. On the other hand, words like *"hospital"* (which translates to hospital) seem to be related to more negative sentiment towards Covid19.

The next figure (Figure 19) show us the comparison between the common words in each label. The upper half figure is the comparison in the Twint dataset, while the second half figure is the comparison in the Kaggle dataset.



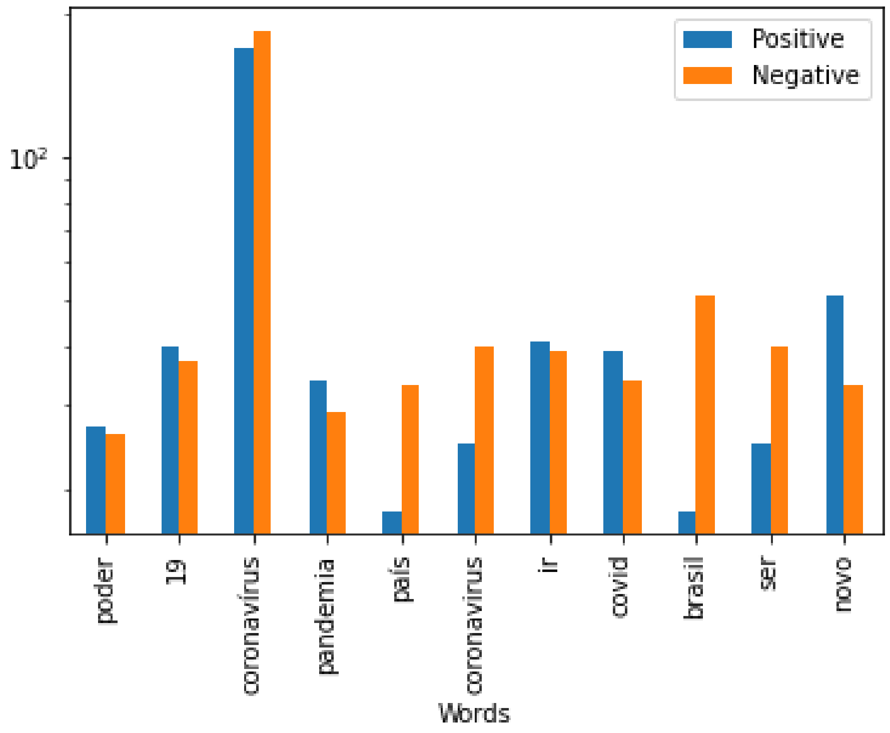


Figure 19: Common terms from both labels

The words present in both datasets, suggests that the common words in both labels are related to our use case. This seems to further indicate that not only the overall dataset is within the Covid topic but each specific label is related to the topic too.

The next figures (Figure 20, Figure 21) show us the most common words on both labels and is a good method to confirm the abovementioned figure. The first Figure (Figure 20) shows us the wordcloud for the Twint dataset, while the second (Figure 21) shows us the wordcloud for the Kaggle dataset.

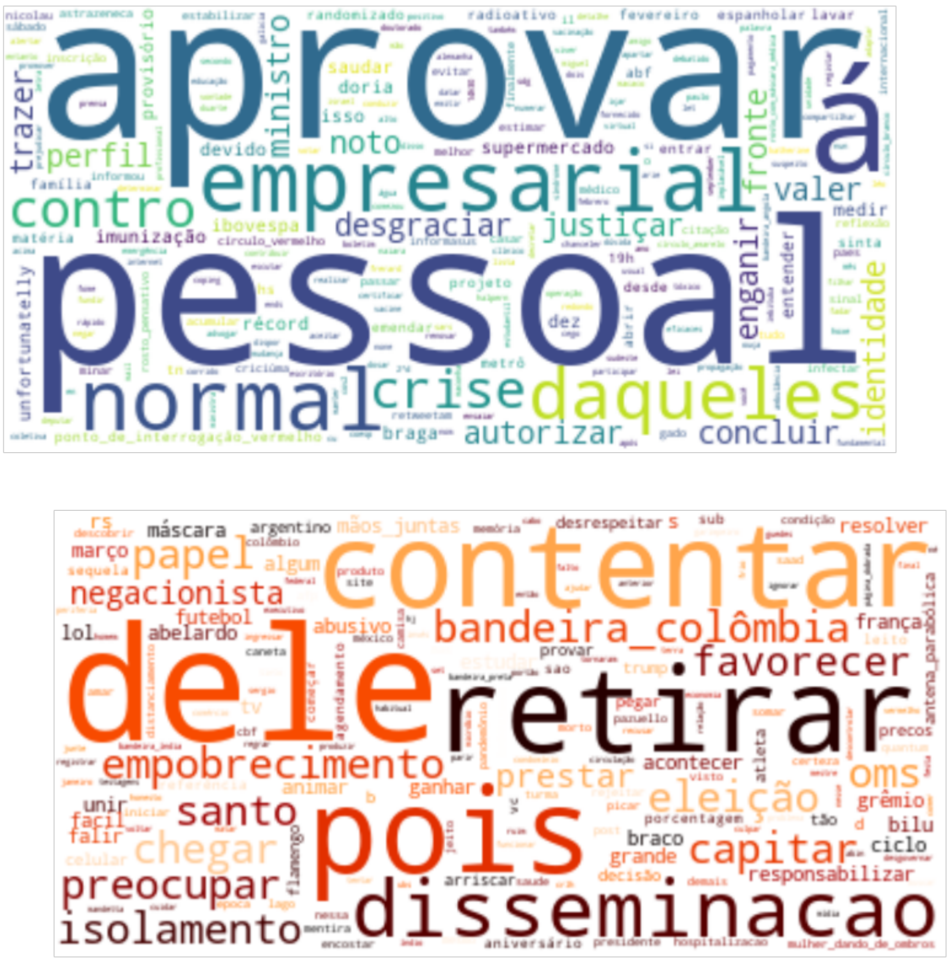


Figure 20: Positive and Negative words Wordcloud for the Twint dataset

The first half of the picture displays, in blueish tones, the most common positive words. This figure (Figure 20) seems to show us that the dataset has good entries for the positive label and our topic and seems to be well labelled. Words like *"imunização"* (which translates to immunization) are a good example of this.

The second half of the figure displays, in reddish tones, the most common negative words. This wordcloud suggests, with words like *"disseminação"*, *"negacionista"* (which translates to dissemination and denialist) among many others, that the dataset has good entries for the negative label and our topic and seems to be well labelled.

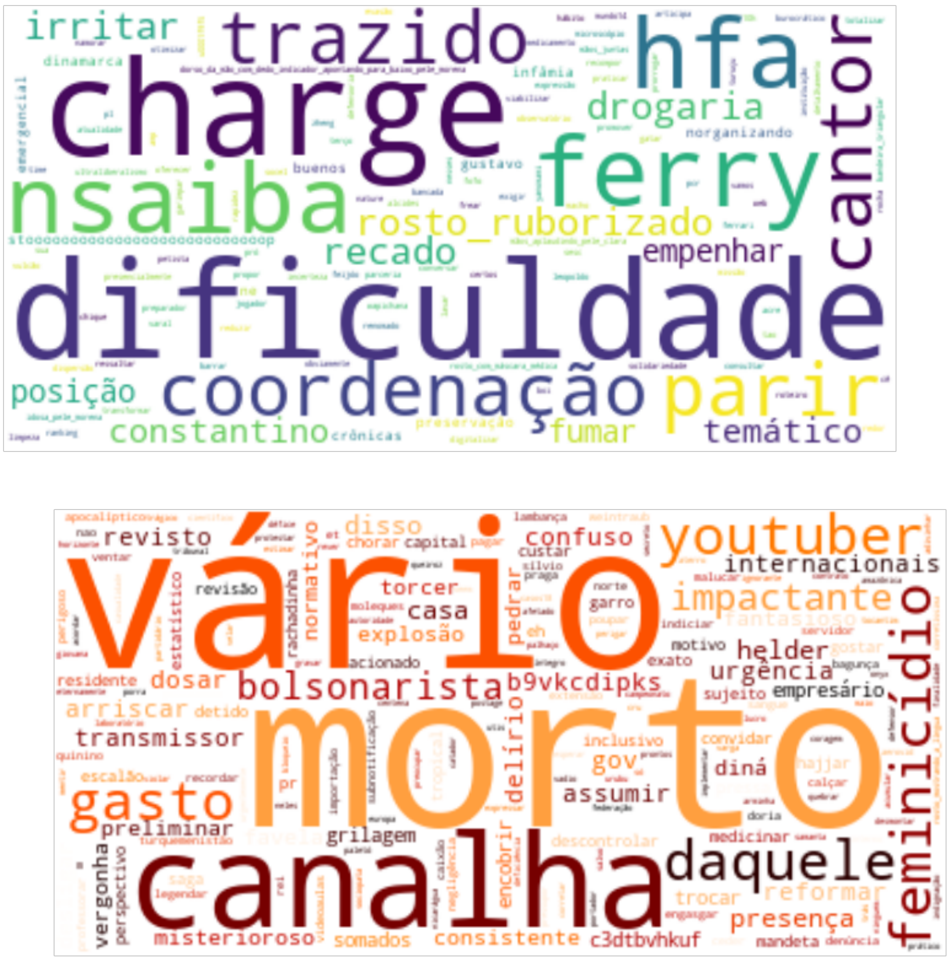


Figure 21: Positive and Negative words Wordcloud for the Kaggle dataset

The first half of the picture displays, in blueish tones, the most common positive words. Words like *"empenhar"* and *"coordenação"* (which translates to effort and coordination) suggest the dataset has good entries for the positive label and our topic and seems to be well labelled.

The second half of the picture displays, in reddish tones, the most common negative words. This wordcloud suggests that the dataset has good entries for the negative label and our topic and seems to be well labelled. Words like *"morto"*, *"canalha"* (which translates to dead and scoundrel) being great examples of words with a negative connotation.

5.3 Transformers

After exploring the datasets, we will create Transformer Pipelines with the purpose of normalising the dataset. These Pipelines will receive the dataset as the input and create a new dataset with the alterations the chosen transformers applied to it. In the next subsection, we will define the pipelines we created and the approach we used to create them.

5.3.1 Transformer Combinations

In this section, we will specify the pipelines created for the normalisation of the dataset. For the creation of these pipelines, we will start with the least complex transformers to the most complex. We have created 6 pipelines.

The first pipeline, pipeline 0, is the baseline pipeline where we do not apply any Transformers to the dataset.

In the second pipeline (Figure 22), stands a pipeline where we normalise the text with simple transformers and remove unnecessary information.

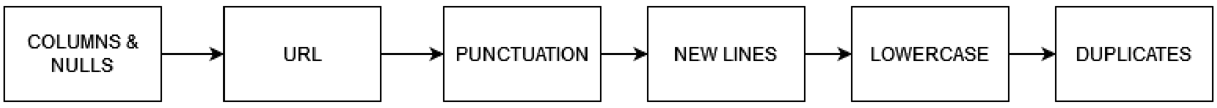


Figure 22: Representation of the transformers used in each step for the pre-processing in Pipeline 1

This pipeline starts by applying the drop nulls and columns Transformer that removes unimportant columns and removes entries with null values. This action is followed by the URL Transformer that removes the links from the text. We continue normalising the text by removing unnecessary punctuation and unnecessary new lines. We end the pipeline by transforming the text into lowercase and removing duplicates.

In the third pipeline (Figure 23), is where we remove unnecessary information from the text and normalise text with a more complex Transformer.

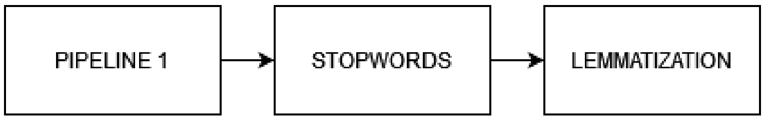


Figure 23: Representation of the transformers used in each step for the pre-processing in Pipeline 2

In this pipeline, we apply Pipeline 1 (Figure 22) followed by the Stopwords Transformers which removes the unnecessary stopwords from the text. After removing stopwords, we normalise the text by tokenizing and lemmatizing the text.

In the fourth pipeline (Figure 24), we normalise text with another new Transformer.

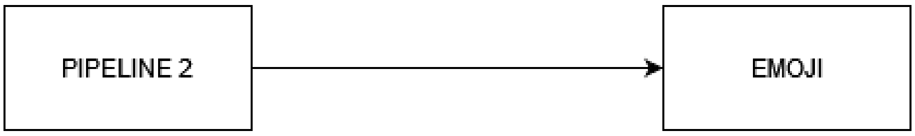


Figure 24: Representation of the transformers used in each step for the pre-processing in Pipeline 3

In this pipeline, we apply Pipeline 2 (Figure 23), followed by the Emoji Transformer that replaces Unicode emojis with a tag with the meaning of the emoji.

In the fifth pipeline (Figure 25), we apply a text normalisation Transformer to the data.

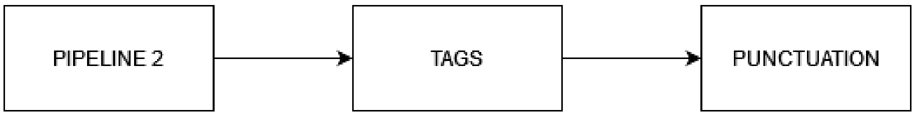


Figure 25: Representation of the transformers used in each step for the pre-processing in Pipeline 4

In this pipeline, we apply Pipeline 2 (Figure 23), followed by the tags Transformer which removes the unnecessary name tags and hashtags from the text. For this pipeline to work properly, we cannot remove punctuation before removing the tags because the Punctuation Transformer removes the # and @ characters that are used in the regular expressions to catch the name tags and hashtags. So the Punctuation Transformer is removed from the Pipeline 1 step and is applied after the Tags Transformers.

In the sixth and last pipeline (Figure 26), applies all the above-mentioned Transformers.

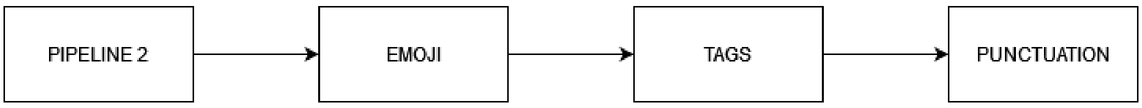


Figure 26: Representation of the transformers used in each step for the pre-processing in Pipeline 5

In this last pipeline, we apply Pipeline 2 (without the Punctuation Transformer) followed by the Emoji Transformer, Tags Transformer and ending with the Punctuation Transformer.

5.4 Models

After having explored the data and having created the Transformer pipelines we are going to apply the normalised datasets to our models. In this section, we will review the results we got for the Kaggle dataset. The results will be split into ML, DL and Autogluon models, and the tests will be standardised in order to compare results.

In the end, we will apply the best pipeline and model combinations in the Twint sample dataset.

5.4.1 Machine Learning

In this section, we will display the results from our ML models. We will standardise the results so that we can compare them. For this, we will have a train-test split of 70/30 for every test performed and in order to diminish the variability of results, we will perform each test 10 times. The result present in the table (Table 6) is the average F1 Score from the 10 tests.

We have the possibility to choose several metrics like accuracy, F1 Score, recall, and precision among others. We chose to use the F1 Score as our metric because it is the harmonic mean of precision and recall, meaning that a good F1 Score indicates good recall and precision.

The dataset is not being changed at all, apart from the Pipeline modifications explained previously.

The results (in percentage, %) for ML are displayed in the following table (Table 6).

Models	Pipeline 0	Pipeline 1	Pipeline 2	Pipeline 3	Pipeline 4	Pipeline 5
Gaussian Naive Bayes	67.1	66.3	67.6	64.7	65.2	68.0
Multinomial Naive Bayes	62.9	68.6	72.7	71.9	71.3	70.6
Random Forest	60.1	73.7	74.0	73.2	73.4	71.8
Bagging Naive Bayes	56.2	63.8	66.6	63.9	66.7	67.4
Bagging Multinomial Naive Bayes	60.7	68.0	71.8	69.4	70.9	69.5

Table 6: Machine Learning results for the Kaggle dataset

The results present in the table above (Table 6) show us that the Random Forest seems to be the best model. Each model seems to handle our pipelines very differently. Still, in most cases, Pipelines 1 and 2 appear to be improving our models while Pipeline 3 appears to be diminishing the score of our models. This may mean that the emoji replacement is not helping us achieve better results with these models, while the removal of stopwords, punctuation and the standardization of the data by lemmatizing it appears to help our model achieve better results.

In general, the results of these models vary according to the pipelines, with the best result achieved being the Random Forest model in Pipeline 2.

From the table we can also, attest that the bagging method, gives us slightly better models in both Gaussian Naive Bayes and Multinomial Naive Bayes.

5.4.2 Deep Learning

For DL models, we will use the exact same standards explained in the previous section, with the addition of a standardised number of epochs that will be set at 25. We will use cross entropy as our loss function. We will use the default hyperparameters for the models used (these values can be found in the documentation of each model).

The results (in percentage, %) for DL are displayed in the following table (Table 7).

Models	Pipeline 0	Pipeline 1	Pipeline 2	Pipeline 3	Pipeline 4	Pipeline 5
CNN + Word2Vec	68.5	73.0	69.1	71.2	73.3	71.7
RNN	76.4	76.3	75.5	78.1	75.0	75.6
LSTM	80.0	89.3	89.3	89.3	89.3	89.3

Table 7: Deep Learning results for the Kaggle dataset

The results offered in the table above (Table 7) indicate that the LSTM is clearly the best model of the three, even though the other two models have results that are better in comparison to the ML models. The LSTM model doesn't seem to be affected in terms of F1 Score by the changes present in our pipelines, apart from Pipeline 1. For RNN and CNN models, we can see that the Pipelines do change the results overall. While in the CNN model, Pipelines 1 and 4 denote an improvement in the overall score, Pipeline 2 states a small deterioration in the results. For the RNN, Pipeline 3 improves our score, while Pipeline 2, 4 and 5 diminish our score.

In general, DL models give a good set of results (in terms of F1 Score), specifically when compared to ML.

5.4.3 Autogluon

For the Autogluon models, we will use the exact hyperparameter and standards as described in the last section. The table below (8) displays the results (in percentage, %) for the Autogluon models.

Models	Pipeline 0	Pipeline 1	Pipeline 2	Pipeline 3	Pipeline 4	Pipeline 5
LightGBM	65.3	67.5	64.1	67.1	69.8	63.4
LightGBM Large	65.9	66.9	62.8	67.6	68.6	61.5
LightGBMXT	64.8	68.4	64.9	66.9	68.9	62.3
CatBoost	59.4	63.5	58.0	59.3	63.4	55.0
XGBoost	65.8	67.6	61.8	65.6	67.0	61.2
WeightedEnsemble	71.6	71.7	68.9	69.9	73.4	68.0
NeuralNetTorch	59.2	66.1	63.1	64.4	67.3	64.0
HFAutoModel	66.1	66.2	66.9	66.4	65.9	65.9
HFAutoModel+Bert	84.4	83.6	76.8	76.7	78.4	78.7

Table 8: Autogluon results for the Kaggle dataset

The results displayed in the table (Table 8) reveal that the HFAutoModel+Bert and the WeightedEnsemble are the best models in terms of F1 Score. In general, for these models Pipelines 1 and 4 gives an improvement in our score, while Pipelines 2 and 5 decrease our score.

From the table, we can also ascertain a major improvement in the HFAutoModel when the Bert embedding was added.

In general, the results obtained with Autogluon models seem to be as good as our Deep Learning results.

5.4.4 Metrics

There are several evaluation metrics that we could have used. For each project, different metrics are more relevant than others. For instance, accuracy is normally used when we need to know the percentage of correct predictions. Recall is normally used when we need to find the percentage of correct positive labels. Precision is used when we want to know how many positive predictions were labelled correctly. F1 Score is commonly used to compare results as it is a combination of the recall and precision results.

In this section, we will test some combinations of models and pipelines with accuracy as its evaluation metric. The following table (Table 9) presents us with these results.

Models	Pipeline	Accuracy (%)
Random Forest	1	76.7
Random Forest	2	68.9
Random Forest	3	68.5
LSTM	3	85.9
LSTM	5	85.7
RNN	3	59.1
HFAutoModel+Bert	0	80.4
HFAutoModel+Bert	1	83.6
WeightedEnsemble	4	69.0
LightGBM	4	65.1
LightGBMXT	4	65.5

Table 9: Accuracy results for the Kaggle dataset

As we can see, with accuracy as our metric, Random Forests are very affected by pipelines 2 and 3. These pipelines make us lose knowledge previously known by the model with Pipeline 1 applied.

The RNN model was a good model for the F1 Score as a metric, while for accuracy, it is a medium model that does not compare with the rest of the models present in the table (Table 9) in terms of accuracy.

The best models are still LSTM and the HFAutoModel+Bert. In LSTMs, we still have a very similar model in terms of its metric. With F1 Score and accuracy, the model does not vary much with the transformations performed by the pipelines. With the HFAutoModel+Bert the inverse happens, the transformations performed by Pipeline 1 increase the quality of our results.

5.4.5 Twint

In this section, we will apply the best combination of models and pipelines from the previous section to the Twint sample dataset. The hyperparameters and standards used in the previous sections will be exactly the same here.

The following table (Table 10) provides us with the results.

Models	Pipeline	F1 Score (%)	Accuracy (%)
Random Forest	1	74.4	63.3
Random Forest	2	76.4	63.4
Random Forest	3	73.5	63.2
LSTM	3	83.7	73.4
LSTM	5	80.7	77.2
RNN	3	75.7	68.1
HFAutoModel+Bert	0	75.4	72.7
HFAutoModel+Bert	1	81.9	77.0
WeightedEnsemble	4	81.2	75.4
LightGBM	4	76.8	68.6
LightGBMXT	4	76.5	68.6

Table 10: Results for the Twint dataset

The results present in the table above (Table 10), are not statistically important due to the low amount of data in the Twint dataset, but still provide us with a pathway to take for other projects where pre-built datasets can not be found. Even though the results are not as important as the results for the Kaggle dataset, we can still visualise that ML models seem to be a bit worse, as expected, when comparing with DL and Autogluon models. The model that seems to be the best is the HFAutoModel with the Bert embedding, being closely followed by the WeightedEnsemble. In general, the results seem to be good, even though we have a very small and slightly unbalanced dataset.

5.4.6 Next Steps

If the results were not satisfactory, there are a couple more steps we could have tried. Firstly, we could have changed the pre-processing. As we can see from our results, some Pipelines work better than others. This may be because some Transformers being used are not appropriate for our dataset, meaning that those transformers eliminate knowledge the model could be capturing. If we look at WeightedEnsemble (Kaggle dataset) we can see that Pipeline 4 is the best pipeline for that model. This may mean that the Tags Transformer is a good pre-processing method for this specific model, while Pipeline 2 and 3 gives us an idea that the emoji, stopwords and lemmatization methods appear to be bad pre-processing methods for this specific model. We could try removing these methods and see the results. We can not expect to get better results, whenever doing this. This simplistic way of thinking can get us better results, sometimes, but ML and DL are not this straightforward.

Secondly, we can optimise our model by changing its hyperparameters. In DL we can try to find the optimal values for the different hyperparameters. We can change some global hyperparameters like the number of epochs for training, batch size, learning rate, optimizer and loss function. For the number of epochs, we can change to the value where the loss function starts to stabilise. We can classify this stabilisation when the loss function has the same value for several epochs in a row. We can change the optimizer being used during the training, the way weights are calculated during the training process may affect the resulting model. There are several more hyperparameters including hyperparameters specific to the model that can be optimised to get a better model. Autogluon also presents the opportunity to optimise hyperparameters.

If these two options, would not grant us the results we want, we may have to check the quality of the dataset itself. The best way to ascertain the quality of the dataset is to manually check its labelling. For larger datasets, this may be prohibitively costly. In these cases, we could try to use pre-trained models or lexicons to evaluate the dataset.

Chapter 6

Conclusions & Future Work

In this chapter, we will delve into the conclusions of this project and the possible future work. We will start with the conclusions followed by the possible future work for the Omnia Package and its integration of this thesis into the Twitter Observatory project.

6.1 Conclusions

The main objective of this thesis was to create a tool that could streamline the process of creating sentiment analysis models for the Portuguese language. This objective was achieved with the development of a set of tools as part of the Omnia Text Mining sub-package. This sub-package lets us create pipelines composed of a reader, that reads the dataset, followed by a set of transformers and models. With this simple pipeline, a user can create SA models in a streamlined way without the know-how to create the models themselves.

To exemplify how we approach a SA problem we created a use case. This use case was Covid19 and we applied this package to datasets related to our use case. Our idea for the use case was to create a dataset and get an already existing dataset to create two different paths a user can do to get data. After getting the data, the package was almost ready to be applied. Before applying the package we did a small exploratory data analysis to ascertain if the datasets were well-labelled and within our topic. After analysing the dataset, we saw that both datasets were within our topic and seemed to be well-labelled. We also noticed that the Twint dataset was slightly unbalanced, which can be a problem in smaller datasets. With the common terms analysis, we were able to ascertain that the most common words for each label seemed to be related to the sentiment present in the label.

After conducting the EDA to the datasets, we created a set of pipelines composed of Transformers.

These pipelines performed several methods of pre-processing and outputted a new dataset with the transformations performed by the methods present in the pipeline.

We applied the Pipelines to the Kaggle dataset, and we used the resulting transformed dataset for the creation of machine learning and deep learning models. We ascertained, for the Kaggle dataset, that the LSTM model seemed to be the standout, even though the other models, like the HFAutoModel with Bert, were close behind.

After discussing the Kaggle results, we took the best pipeline combination and applied it to our Twint sample dataset where we ascertained that the HFAutoModel with the Bert embedding is the best model followed closely behind by the WeightedEnsemble. The results for this dataset were good, even though they are not statistically important due to the low amount of data. Despite this, the Twint dataset clearly showcases a path for a creation of a dataset that can be applied to our project.

Overall the results were good. We can hypothesise that Autogluon and Deep Learning are less affected by Pipeline transformations in comparison to Machine Learning. In general, different models were better accompanied by different pipelines, which makes us hypothesise that there is no pre-processing Pipeline that can be used every time for these datasets. From the results using Bert, we can infer that adding embeddings to models seems to better their results.

In general, these two datasets showed the potential this package has for TM and SA in Portuguese. This package, helps us create Pipelines of pre-processing methods, Transformers, to apply to datasets. We can also create several Models and compare their results with several metrics.

This can all be streamlined by joining several models to the Transformer Pipeline and creating all the models for that pipeline. We can later analyse the results from the different models to choose the best.

In the next section, we will delve into what possible next steps can be done for this project.

6.2 Future Work

The Omnia package, especially the Text Mining module, can be improved by adding more pre-processing methods (Transformers). Transformers like Accent Removal, Contraction expansion, Roman Numerals Replacement, among many others, may be good Transformers to add. Another addition, that the Omnia package could include, is the addition of more models. In terms of DL, the Omnia package is still lacking and the addition of more models could be interesting. The addition of embeddings like Bert or Word2vec are good additions to the package.

After enhancing the Transformers and Models of the package we can start thinking of more complex problems. Problems like the automatic labelling of the datasets can be a good addition to the package. Multi-lingual support is another problem that can be tackled, this problem can be simply solved by using translation. But this solution is not always the best.

Another step to improve this project could be the creation of a couple of modules for the Twitter Observatory Web Application. The Twitter Observatory has been started by Constança Elias for her dissertation

[134] and is a Web Application that has the capability of creating datasets for any given topic. By adding a labelling module, this project could label datasets in accordance with the sentiment prevalent in texts. A good solution to this problem is to interactively classify the texts by creating a tool similar to [142], in this project social media posts are classified as relevant or irrelevant (like in the Twitter Observatory) and if it is deemed as relevant the post is approved to an interface where its users will evaluate it in accordance with the Sentiment present in the post. With the utilisation of the Omnia project, we could create ML, DL and Autogluon models for the abovementioned labelled datasets.

This would make the Twitter Observatory a powerful tool for SA, by helping to streamline the creation of SA models without the need for any technological knowledge.

Bibliography

[1] M. Mohammed, M. Khan, and E. Bashier. *Machine Learning: Algorithms and Applications*. July 2016. isbn: 9781498705387. doi: 10.1201/9781315371658.

[2] S. Fentie, A. Demessie, and B. Das Mohapatra. "A Comparative Study on Performance Evalution of Eager versus Lazy Learning Methods". In: (Mar. 2014).

[3] *Supervised, Unsupervised, And Semi-Supervised Learning With Real-Life Usecase*. url: https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning/.

[4] M. Mohri, A. Rostamizadeh, and A. S. Talwalkar. "Foundations of Machine Learning". In: *Adaptive computation and machine learning*. 2012.

[5] N. K. Amandeep Kaur Mann. "Review Paper on Clustering Techniques". In: *Global Journal of Computer Science and Technology* (2013). issn: 0975-4172. url: https://computerresearch.org/index.php/computer/article/view/353.

[6] H. Shinde and A. Sankhe. "Comparison of Enhanced DBSCAN Algorithms: A Review". In: *International journal of engineering research and technology* 5 (2018).

[7] P. Dubey and A. Rajavat. "Comparative Study Between Density Based Clustering - DBSCAN and Optics". In: *International Journal of Advance Computational Engineering and Networking (IJACEN)* 4.12 (2016), pp. 34–37.

[8] S. S. Mary and T. Selvi. "A Study of K-Means and Cure Clustering Algorithms". In: *International Journal of Engineering Research & Technology (IJERT)* (2014), pp. 1–3.

[9] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: an efficient data clustering method for very large databases". In: *In Proc. of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD*. 1996, pp. 103–114.

[10] S. Shukla. "A Review ON K-means DATA Clustering APPROACH". In: 2014.

[11] M. K. Goyal and S. Aggarwal. "A REVIEW ON K-MODE CLUSTERING ALGORITHM". In: *International Journal of Advanced Research in Computer Science* 8 (2017), pp. 725–729.

[12] R. Ng and J. Han. “CLARANS: a method for clustering objects for spatial data mining”. In: *IEEE Transactions on Knowledge and Data Engineering* 14.5 (2002), pp. 1003–1016. doi: 10.1109/TKDE.2002.1033770.

[13] W. Wang, J. Yang, and R. R. Muntz. “STING: A Statistical Information Grid Approach to Spatial Data Mining”. In: *VLDB*. 1997.

[14] R. Agrawal et al. “Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications”. In: *SIGMOD Rec.* 27.2 (June 1998), pp. 94–105. issn: 0163-5808. doi: 10.1145/276305.276314. url: https://doi.org/10.1145/276305.276314.

[15] R. Diwate. “Data Mining Techniques in Association Rule : A Review”. In: (Jan. 2014).

[16] J. Yabing. “Research of an Improved Apriori Algorithm in Data Mining Association Rules”. In: *International Journal of Computer and Communication Engineering* (2013), pp. 25–27.

[17] L. Schmidt-Thieme. “Algorithmic Features of Eclat.” In: (Jan. 2004).

[18] A. Kaur and G. Jagdev. “Analyzing Working of FP-Growth Algorithm for Frequent Pattern Mining”. In: (2017).

[19] GeeksforGeeks. *Introduction to Dimensionality Reduction*. Sept. 2022. url: https://www.geeksforgeeks.org/dimensionality-reduction/.

[20] I. T. Jolliffe and J. Cadima. “Principal component analysis: a review and recent developments”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374 (2016).

[21] D. M. Blei, A. Y. Ng, and M. I. Jordan. “Latent Dirichlet Allocation”. In: *J. Mach. Learn. Res.* 3.null (Mar. 2003), pp. 993–1022. issn: 1532-4435.

[22] *Reinforcement Learning Algorithms and Applications*. url: https://techvidvan.com/tutorials/reinforcement-learning/.

[23] M. van Otterlo and M. A. Wiering. “Markov Decision Processes: Concepts and Algorithms”. In: (2012).

[24] J. W. Ling Wang Zixiao Pan. “A Review of Reinforcement Learning Based Intelligent Optimization for Manufacturing Scheduling”. In: *Complex System Modeling and Simulation* 1.4 (2022), p. 14. doi: 10.23919/CSMS.2021.0027. url: https://www.sciopen.com/article/10.23919/CSMS.2021.0027.

[25] A. K. et al. “Stabilizing Off-Polic Q-Learning via Bootstrapping Error Reduction”. In: (2019). url: http://arxiv.org/abs/1906.00949.

[26] V. P. Malikireddy and M. Kasa. “Customer Churns Prediction Model Based on Machine Learning Techniques: A Systematic Review”. In: *Proceedings of the 3rd International Conference on Integrated Intelligent Computing Communication & Security (ICIIC 2021)* (2021).

[27] L. De Angelis. *Predicting Customer Lifetime Value with “Buy ‘Til You Die” probabilistic models in Python*. 2019. url: https://towardsdatascience.com/predicting-customer-lifetime-value-with-buy-til-you-die-probabilistic-models-in-python-f5cac78758d9.

[28] M. Bohanec, M. Borstnar, and M. Robnik-Sikonja. “Explaining machine learning models in sales predictions”. In: *Expert Systems with Applications* 71 (Apr. 2017), pp. 416–428. doi: 10.1016/j.eswa.2016.11.010.

[29] J. Breitenbach et al. “A Systematic Literature Review of Machine Learning Tools for Supporting Supply Chain Management in the Manufacturing Environment”. In: (Dec. 2021), pp. 2875–2883. doi: 10.1109/BigData52589.2021.9672013.

[30] P. Raghavan and N. Gayar. “Fraud Detection using Machine Learning and Deep Learning”. In: (Dec. 2019), pp. 334–339. doi: 10.1109/ICCIKE47802.2019.9004231.

[31] S. Ahmad et al. “Unsupervised real-time anomaly detection for streaming data”. In: *Neurocomputing* 262 (June 2017). doi: 10.1016/j.neucom.2017.04.070.

[32] J. Nurma Sari et al. “Review on Customer Segmentation Technique on Ecommerce”. In: *Advanced Science Letters* 22 (Oct. 2016), pp. 3018–3022. doi: 10.1166/asl.2016.7985.

[33] S. Gupta. “A Literature Review on Recommendation Systems”. In: *International Research Journal of Engineering and Technology (IRJET)* 7.9 (2020).

[34] D. Silver et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144. doi: 10.1126/science.aar6404. eprint: https://www.science.org/doi/pdf/10.1126/science.aar6404. url: https://www.science.org/doi/abs/10.1126/science.aar6404.

[35] M. van der Ree and M. Wiering. “Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play”. In: *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. 2013, pp. 108–115. doi: 10.1109/ADPRL.2013.6614996.

[36] *quotienthealth.com i*. url: http://www.quotienthealth.com.

[37] 2022. url: https://www.kensci.com/.

[38] Y. LeCun, Y. Bengio, and G. Hinton. “Deep Learning”. In: *Nature* 521 (May 2015), pp. 436–44. doi: 10.1038/nature14539.

[39] Heartfield. *Cloud-Based Cyber-Physical Intrusion Detection for Vehicles Using Deep Learning*. url: https://www.researchgate.net/profile/Ryan-Heartfield/publication/321341597/figure/fig5/AS:667675554480148@1536197665532/MLP-deep-learning-architecture.ppm.

[40] Sonali, B. Maind, and P. Wankar. “Research Paper on Basic of Artificial Neural Network”. In: 2014.

[41] M. Sazli. “A brief review of feed-forward neural networks”. In: *Communications, Faculty Of Science, University of Ankara* 50 (Jan. 2006), pp. 11–17. doi: 10.1501/0003168.

[42] A. Alihodzic. “Training Feed-Forward Neural Networks Employing Improved Bat Algorithm for Digital Image Compression”. In: (2018). Ed. by I. Lirkov and S. Margenov, pp. 315–323.

[43] A. Iqbal and S. Aftab. “A Feed-Forward and Pattern Recognition ANN Model for Network Intrusion Detection”. In: *International Journal of Computer Network and Information Security* 11 (Apr. 2019), pp. 19–25. doi: 10.5815/ijcnis.2019.04.03.

[44] J. Chai et al. “Deep learning in computer vision: A critical review of emerging techniques and application scenarios”. In: *Machine Learning with Applications* 6 (Aug. 2021), p. 100134. doi: 10.1016/j.mlwa.2021.100134.

[45] Y. Wang et al. “Passive Sonar Target Tracking Based on Deep Learning”. In: *Journal of Marine Science and Engineering* 10.2 (2022). issn: 2077-1312. doi: 10.3390/jmse10020181. url: https://www.mdpi.com/2077-1312/10/2/181.

[46] G. Hinton et al. “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. doi: 10.1109/MSP.2012.2205597.

[47] W. Zhu. “Classification of MNIST Handwritten Digit Database using Neural Network”. In: (2018).

[48] Z. Lipton. “A Critical Review of Recurrent Neural Networks for Sequence Learning”. In: (May 2015).

[49] P. . K. . Anwla. *Recurrent Neural Network (RNN) architecture explained in detail*. Apr. 2022. url: https://towardsmachinelearning.org/recurrent-neural-network-architecture-explained-in-detail/.

[50] C. Sousa. “Analysis of the backpropagation algorithm using linear algebra”. In: June 2012, pp. 1–8. isbn: 978-1-4673-1488-6. doi: 10.1109/IJCNN.2012.6252364.

[51] G. Van Houdt, C. Mosquera, and G. Nápoles. “A Review on the Long Short-Term Memory Model”. In: *Artificial Intelligence Review* 53 (Dec. 2020). doi: 10.1007/s10462-020-09838-1.

[52] J. Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: (Dec. 2014).

[53] Y. Ning et al. “A Review of Deep Learning Based Speech Synthesis”. In: *Applied Sciences* 9.19 (2019). issn: 2076-3417. doi: 10.3390/app9194050. url: https://www.mdpi.com/2076-3417/9/19/4050.

[54] A.-I. Marinescu. “Bach 2.0 - generating classical music using recurrent neural networks”. In: *Procedia Computer Science* 159 (Jan. 2019), pp. 117–124. doi: 10.1016/j.procs.2019.09.166.

[55] K. O’Shea and R. Nash. “An Introduction to Convolutional Neural Networks”. In: *ArXiv e-prints* (Nov. 2015).

[56] A. V. S M. “TRAFFIC SIGN RECOGNITION AND CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS”. In: *Journal of Emerging Technologies and Innovative Research* 6.2 (2019). url: https://www.jetir.org/papers/JETIRAB06034.pdf.

[57] T. Hossain et al. “Brain Tumor Detection Using Convolutional Neural Network”. In: *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*. 2019, pp. 1–6. doi: 10.1109/ICASERT.2019.8934561.

[58] L. Sadouk. “CNN Approaches for Time Series Classification”. In: *Time Series Analysis - Data, Methods, and Applications* (2019).

[59] G. Montufar. “Restricted Boltzmann Machines: Introduction and Review”. In: (June 2018).

[60] L. Gutiérrez and B. Keith Norambuena. “A Systematic Literature Review on Word Embeddings: Proceedings of the 7th International Conference on Software Process Improvement (CIMPS 2018)”. In: (Jan. 2019), pp. 132–141. doi: 10.1007/978-3-030-01171-0_12.

[61] T. Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *ICLR*. 2013.

[62] S. Pan and T. Ding. “Social Media-based User Embedding: A Literature Review”. In: *CoRR* abs/1907.00725 (2019). arXiv: 1907.00725. url: http://arxiv.org/abs/1907.00725.

[63] S. France and J. Carroll. “Two-Way Multidimensional Scaling: A Review”. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 41 (Oct. 2011), pp. 644–661. doi: 10.1109/TSMCC.2010.2078502.

[64] A. Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. url: http://arxiv.org/abs/1706.03762.

[65] J. Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[66] M. Arifuzzaman and E. Arslan. “Learning Transfers via Transfer Learning”. In: (Nov. 2021). doi: 10.1109/INDIS54524.2021.00009.

[67] H.-D. Wehle. “Machine Learning, Deep Learning, and AI: What’s the Difference?” In: July 2017.

[68] M. Ahmed. *UK passport photo checker shows bias against dark-skinned women*. 2022. url: https://www.bbc.com/news/technology-54349538.

[69] A. Singhal and I. Google. “Modern Information Retrieval: A Brief Overview”. In: *IEEE Data Engineering Bulletin* 24 (Jan. 2001).

[70] K. Jayaram and K. Sangeeta. "A review: Information extraction techniques from research papers". In: *2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*. 2017, pp. 56–59. doi: 10.1109/ICIMIA.2017.7975532.

[71] S. Joseph et al. "Natural Language Processing: A Review". In: *Natural Language Processing: A Review* 6 (Mar. 2016), pp. 207–210.

[72] T. Vollmer. *The Limits of Copyright: Text and Data Mining - Creative Commons*. 2022. url: https://creativecommons.org/2015/01/21/the-limits-of-copyright-text-and-data-mining/.

[73] B. Saju, S. Jose, and A. Antony. "Comprehensive Study on Sentiment Analysis: Types, Approaches, Recent Applications, Tools and APIs". In: (2020), pp. 186–193. doi: 10.1109/ACCTHPA49271.2020.9213209.

[74] Y. A. Solangi et al. "Review on Natural Language Processing (NLP) and Its Toolkits for Opinion Mining and Sentiment Analysis". In: (2018), pp. 1–4. doi: 10.1109/ICETAS.2018.8629198.

[75] Q. Zhang et al. "Text mining and sentiment analysis of COVID-19 tweets". In: (June 2021).

[76] X. Zhang and X. Zheng. "Comparison of Text Sentiment Analysis Based on Machine Learning". In: (2016), pp. 230–233. doi: 10.1109/ISPDC.2016.39.

[77] H. Zarzour et al. "Sentiment Analysis Based on Deep Learning Methods for Explainable Recommendations with Reviews". In: (2021), pp. 452–456. doi: 10.1109/ICICS52457.2021.9464601.

[78] H. Chen et al. "Fine-grained Sentiment Analysis of Chinese Reviews Using LSTM Network". In: *Journal of Engineering Science and Technology Review* 11 (Feb. 2018), pp. 174–179. doi: 10.25103/jestr.111.21.

[79] A. Khattak et al. "Fine-Grained Sentiment Analysis for Measuring Customer Satisfaction Using an Extended Set of Fuzzy Linguistic Hedges". In: *International Journal of Computational Intelligence Systems* 13 (1 2020), pp. 744–756. issn: 1875-6883. doi: https://doi.org/10.2991/ijcis.d.200513.001. url: https://doi.org/10.2991/ijcis.d.200513.001.

[80] Y. Ding et al. "Selective Sparse Sampling for Fine-Grained Image Recognition". In: (2019), pp. 6598–6607. doi: 10.1109/ICCV.2019.00670.

[81] V. Gajarla. "Emotion Detection and Sentiment Analysis of Images". In.

[82] A. S. Imran et al. "Cross-Cultural Polarity and Emotion Detection Using Sentiment Analysis and Deep Learning on COVID-19 Related Tweets". In: *IEEE Access* 8 (2020), pp. 181074–181090. doi: 10.1109/ACCESS.2020.3027350.

[83] A. Cahyadi and M. L. Khodra. “Aspect-Based Sentiment Analysis Using Convolutional Neural Network and Bidirectional Long Short-Term Memory”. In: *2018 5th International Conference on Advanced Informatics: Concept Theory and Applications (ICAICTA)*. 2018, pp. 124–129. doi: 10.1109/ICAICTA.2018.8541300.

[84] S. Gojali and M. L. Khodra. “Aspect based sentiment analysis for review rating prediction”. In: *2016 International Conference On Advanced Informatics: Concepts, Theory And Application (ICAICTA)*. 2016, pp. 1–6. doi: 10.1109/ICAICTA.2016.7803110.

[85] N. X. Bach, L. C. Linh, and T. M. Phuong. “Cross-Domain Intention Detection in Discussion Forums”. In: SoICT 2017. Nha Trang City, Viet Nam: Association for Computing Machinery, 2017, pp. 173–180. isbn: 9781450353281. doi: 10.1145/3155133.3155182. url: https://doi.org/10.1145/3155133.3155182.

[86] C. Hutto and E. Gilbert. *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*. Jan. 2015.

[87] S. Loria et al. *TextBlob: Simplified Text Processing Documentation*.

[88] A. Esuli and F. Sebastiani. *SENTIWORDNET: A Publicly Available Lexical Resource for Opinion Mining*. 2006.

[89] J. J. Almeida. *Projecto Natura*. url: https://natura.di.uminho.pt/wiki/doku.php.

[90] *NetLang Project*. url: https://sites.google.com/site/projectnetlang/introduction.

[91] J. B. Gonçalves et al. “NetAC, An Automatic Classifier of Online Hate Speech Comments”. In: *Trends and Applications in Information Systems and Technologies*. Ed. by Á. Rocha et al. Cham: Springer International Publishing, 2021, pp. 494–505. isbn: 978-3-030-72660-7. doi: https://doi.org/10.1007/978-3-030-72660-7_47.

[92] E. Santos Duarte. *Sentiment Analysis on Twitter for the Portuguese Language*. 2013. url: https://run.unl.pt/bitstream/10362/11338/1/Duarte_2013.pdf.

[93] N. S. Hartmann et al. “Portuguese Word Embeddings: Evaluating on Word Analogies and Natural Language Tasks”. In: (2017).

[94] *Pre-trained Portuguese BERT models •*. Mar. 2020. url: https://opencor.gitlab.io/corpora/souza20pre/.

[95] R. Rodrigues et al. “Multilingual Transformer Ensembles for Portuguese Natural Language Tasks”. In: (Mar. 2020).

[96] jneto04. url: https://github.com/jneto04/ner-pt.

[97] J. W. Pennebaker et al. “The Development and Psychometric Properties of LIWC2015”. In: (2015).

[98] M. M. Bradley and P. J. Lang. “Affective Norms for English Words (ANEW): Instruction Manual and Affective Ratings”. In: (1999).

[99] S. M. Mohammad and P. D. Turney. "Crowdsourcing a Word-Emotion Association Lexicon". In: *Computational Intelligence* 29.3 (2013), pp. 436–465.

[100] F. Nielsen. "A new ANEW: Evaluation of a word list for sentiment analysis in microblogs". In: *CoRR* (Mar. 2011).

[101] M. Hu and B. Liu. "Mining and Summarizing Customer Reviews". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04. Seattle, WA, USA: ACM, 2004, pp. 168–177. isbn: 1-58113-888-1. doi: 10.1145/1014052.1014073. url: http://doi.acm.org/10.1145/1014052.1014073.

[102] T. Loughran and B. McDonald. "When Is a Liability Not a Liability? Textual Analysis, Dictionaries, and 10-Ks". In: *The Journal of Finance* 66.1 (2011), pp. 35–65. doi: 10.1111/j.1540-6261.2010.01625.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.2010.01625.x. url: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.2010.01625.x.

[103] E. Cambria et al. "Sentic Computing: Exploitation of Common Sense for the Development of Emotion-Sensitive Systems". In: Jan. 2009, pp. 148–156. isbn: 978-3-642-12396-2. doi: 10.1007/978-3-642-12397-9_12.

[104] M. Taboada and J. Grieve. "Analysing Appraisal automatically". In: *In Proceeding of AAAI spring symposium on exploring attitude and affect in text* (Apr. 2004), pp. 158–161.

[105] A. Kilgarriff. "SENSEVAL: An exercise in evaluating word sense disambiguation programs". In: (Feb. 1999).

[106] M. Learn. *Sentiment Analysis: A Definitive Guide*. url: https://monkeylearn.com/sentiment-analysis/.

[107] Rosette. *rosette text analytics platform - ai for human language*. url: https://www.rosette.com/.

[108] *the news intelligence platform - aylien news api*. url: https://aylien.com/.

[109] J. Pennington, R. Socher, and C. D. Manning. "GloVe: Global Vectors for Word Representation". In: *EMNLP*. 2014.

[110] R. Collobert et al. "Natural Language Processing (Almost) from Scratch". In: *J. Mach. Learn. Res.* 12 (2011), pp. 2493–2537.

[111] D. Tang et al. "Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification". In: *ACL*. 2014.

[112] M. E. Peters et al. *Deep contextualized word representations*. 2018. arXiv: 1802.05365 [cs.CL].

[113] A. Akbik, D. Blythe, and R. Vollgraf. “Contextual String Embeddings for Sequence Labeling”. In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 1638–1649. url: https://aclanthology.org/C18-1139.

[114] A. Akbik. *flair/TUTORIAL_4_ELMO_BERT_FLAIR_EMBEDDING.md at master · flairNLPflair*. url: https://github.com/flairNLP/flair/blob/master/resources/docs/TUTORIAL_4_ELMO_BERT_FLAIR_EMBEDDING.md.

[115] *NumPy*. url: https://numpy.org/.

[116] *emoji*. Sept. 2022. url: https://pypi.org/project/emoji/.

[117] *spaCy · Industrial-strength Natural Language Processing in Python*. url: https://spacy.io/.

[118] *NLTK :: Natural Language Toolkit*. url: https://www.nltk.org/.

[119] *AutoGluon: AutoML for Text, Image, and Tabular Data — AutoGluon Documentation 0.5.2 documentation*. url: https://auto.gluon.ai/stable/index.html.

[120] *PyTorch*. url: https://pytorch.org/.

[121] *TensorFlow: Create production-grade machine learning models with TensorFlow*. url: https://www.tensorflow.org/.

[122] *Keras: the Python deep learning API*. url: https://keras.io/.

[123] *scikit-learn: machine learning in Python — scikit-learn 1.1.2 documentation*. url: https://scikit-learn.org/stable/index.html.

[124] *pandas - Python Data Analysis Library*. url: https://pandas.pydata.org/.

[125] *re — Regular expression operations — Python 3.10.8 documentation*. url: https://docs.python.org/3/library/re.html.

[126] *googletrans*. June 2020. url: https://pypi.org/project/googletrans/.

[127] *OpenNMT - Open-Source Neural Machine Translation*. url: https://opennmt.net/.

[128] *twint*. Apr. 2020. url: https://pypi.org/project/twint/.

[129] *Matplotlib — Visualization with Python*. url: https://matplotlib.org/.

[130] *Kaggle: Your Machine Learning and Data Science Community*. url: https://www.kaggle.com/.

[131] *DrivenData:Data science competitions to build a better world*. url: https://www.drivendata.org/.

[132] *The CrowdANALYTIX Community: where data experts collaborate compete to build optimize AI, ML, NLP and Deep Learning algorithms*. url: https://www.crowdanalytix.com/community.

[133] *AIcrowd*. url: https://www.aicrowd.com/crowdai.html?rd=/.

[134] C. Elias and M. Rocha. “Twitter Observatory: developing tools to recover and classify information for the social network Twitter”. In: (Oct. 2022).

[135] *translators*. Oct. 2022. url: https://pypi.org/project/translators/.

[136] *string — Common string operations — Python 3.10.8 documentation*. url: https://docs.python.org/3/library/string.html.

[137] Baziotis. *GitHub - cbaziotis/ekphrasis: Ekphrasis is a text processing tool, geared towards text from social networks, such as Twitter or Facebook. Ekphrasis performs tokenization, word normalization, word segmentation (for splitting hashtags) and spell correction, using word statistics from 2 big corpora (english Wikipedia, twitter - 330mil english tweets).* url: https://github.com/cbaziotis/ekphrasis.

[138] R. Rodrigues et al. “Zero-shot hashtag segmentation for multilingual sentiment analysis”. In: (Dec. 2021).

[139] *Hugging Face – The AI community building the future.* url: https://huggingface.co/.

[140] *Coronavirus Twitters NLP: Text Classification (PT)*. July 2020. url: https://www.kaggle.com/datasets/francielleavargas/opcovidbr.

[141] F. A. Vargas, R. S. S. D. Santos, and P. R. Rocha. “Identifying fine-grained opinion and classifying polarity of twitter data on coronavirus pandemic”. In: (2020), pp. 01–10.

[142] L. Snyder et al. “Interactive Learning for Identifying Relevant Tweets to Support Real-time Situational Awareness”. In: (Aug. 2019).